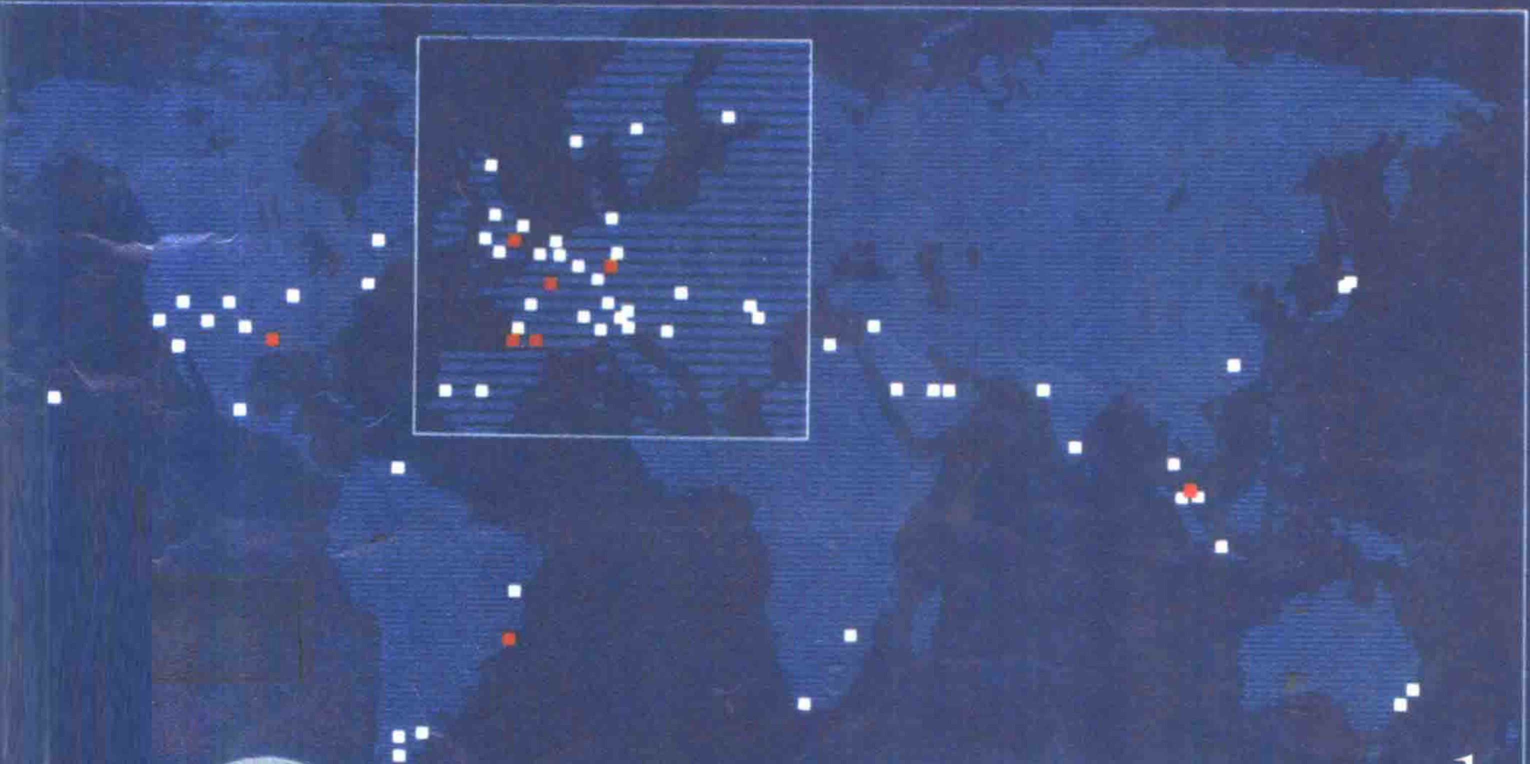
HOPE

Turbo Basic 教程

凯文　文山 编译

（共二册）



北京希望电脑公司

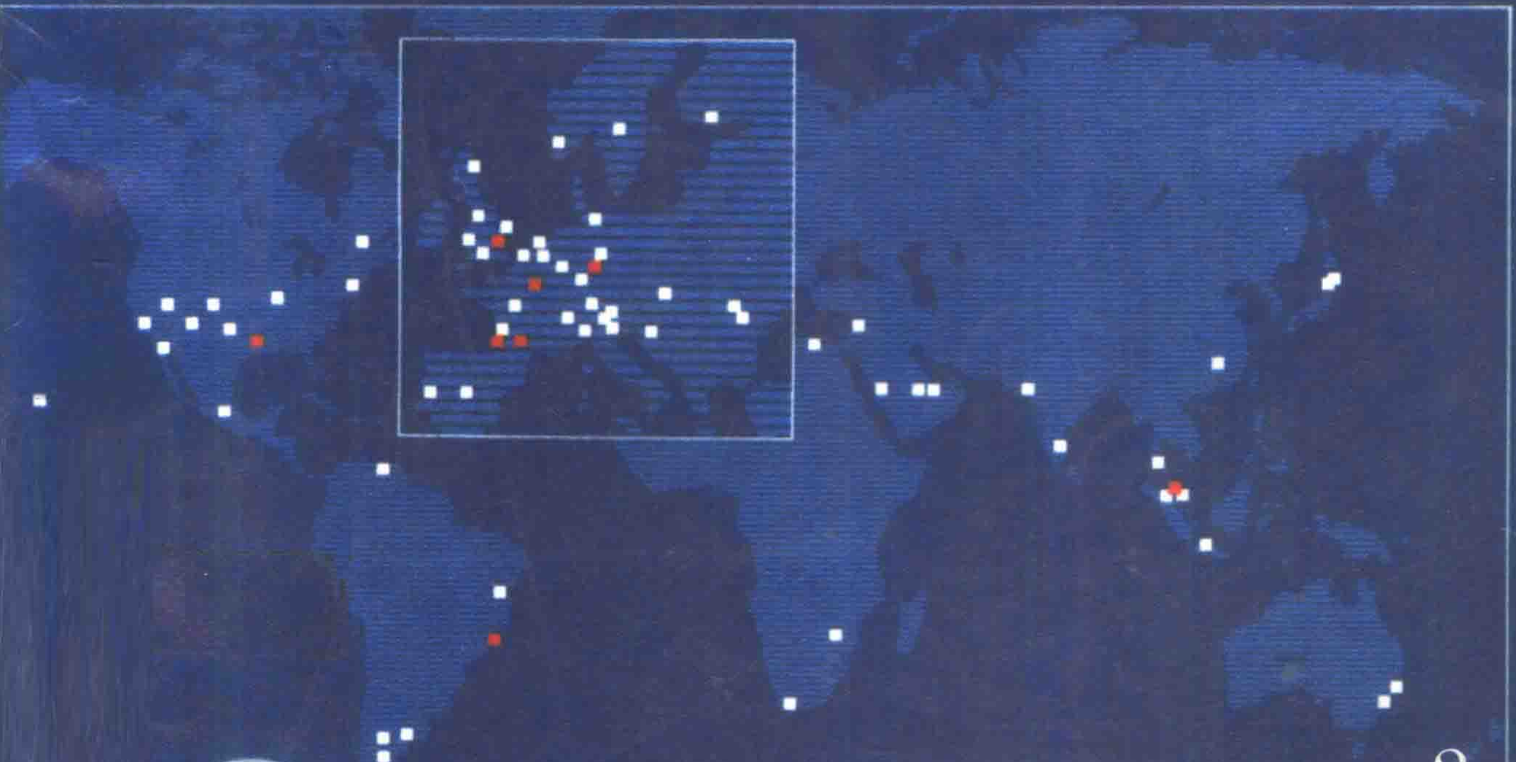
1

HOPE

Turbo Basic 教程

凯文　文山 编译

（共二册）



北京希望电脑公司

2

Turbo Basic 教程

（上 册）

凯 文　文 山　编译

北京希望电脑公司

一九九一年十一月

Turbo Basic 教程

（下　册）

凯 文　文 山　编译

北京希望电脑公司

一九九一年十一月

前　言

Basic语言是一种普及型、易学的程序设计语言．它已广泛应用于工业、农业、商业、科研、教学等方面，受到广大计算机使用人员的厚爱．据美国有关方面的统计资料报道，Basic语言是目前美国使用最广泛的计算机语言之一；在波士顿的程序员协会中，80%以上的会员使用Basic语言．

Basic语言从最初的解释型语言发展到目前版本达几十种的编译型语言，功能已得到极大的增强，语句更加丰富．目前流行较广的有：Basic、BASICA、GWBASIC、QuikyBasic、Ture Basic和Turbo Basic等；其中Turbo Basic因它的良好结构化、功能齐全而又简单易学等优点独树一帜，受到广大计算机用户的欢迎．Turbo Basic语言与简单Basic的区别除了它的结构化好外，还包括如下方面:

- ☐ 提供上下文有关的帮助
- ☐ 完全支持递归
- ☐ 具备定制用户界面和编辑器的能力
- ☐ 为字符串提供充足的64K内存
- ☐ 标准的IEEE浮点格式
- ☐ 支持8087 (数学协处理器)
- ☐ 程序的大小只受计算机保用内存的限制（而不是64K的限制）
- ☐ 支持EGA和CGA
- ☐ 支持IMB PS／2和MCGQ；在640K×480 的分辨率下，可以使用2种和16种颜色．
- ☐ 编译器、编辑器、运行时程序库和可执行程序的完全集成；具有独立的编辑和跟踪信息、连接程序库、用户界面及执行窗口．
- ☐ 发生编译、运行和I／O错误时，在源程序中指出错误发生处．
- ☐ 可访问局部、静态和全局变量
- ☐ 新的长整型（32位）
- ☐ 下拉选单
- ☐ 全窗口管理

本书基于最新的Turbo Basic 2.0版，全面而系统地介绍了Turbo Basic的使用方法及其功能、特性．本书始终贯彻结构化的思想，用软件工程的方法设计、分析各种实用问题．书中提供了大量的实例和自我测验题、复习题、上机调试题等，并附有答案，因此它是一种实用的Turbo Basic教材，适合所有初学者、大学生以及想深入了解和使用Turbo Basic的科学技术人员使用．

本书共分两册，由六部分组成(含十四章和六个附录).

上册：概述Turbo Basic程序设计的概念及其方法，并细述Turbo Basic的逻辑控制结构和打印技巧；此外，还介绍了Turbo Basic程序的调试方法。本册是Turbo Basic的基础部分，由第一至五部分所组成，具体包括：

第 一 章　结构化程序设计的概念
第 二 章　Turbo Basic结构化程序设计
第 三 章　输入／处理／输出：编写基本程序
第 四 章　IF选择语句
第 五 章　关于逻辑控制及自顶而下编程的迭代、循环和子程序
第 六 章　利用输入文件或键入数据显示和打印设计良好的报表
第 七 章　控制中断过程：自顶向下模块化方法
第 八 章　调试程序、确认数据和改善程序设计
第 九 章　深入了解Turbo Basic环境

下册：介绍Turbo Basic高级特性中的数组、表格处理、文件处理、数据库管理、图形和声音等，并对自顶向下模块化程序设计方法进行详细讨论，本册属Turbo Basic的高深部分；此外，为方便读者使用，本册还提供了六个附录。具体包括：

第 十 章　数组及表格处理
第十一章　用过程进行自顶向下模块化程序设计详述
第十二章　自顶向下模块化程序设计中的函数、字符串处理及递归
第十三章　文件处理和数据库管理
第十四章　图形和声音
附 录 A　Turbo Basic保留字
附 录 B　Turbo Basic字符集
附 录 C　ASCII(美国信息交换标准代码)表
附 录 D　Turbo Basic编辑命令表
附 录 E　Turbo Basic错误信息
附 录 F　Turbo Basic指令集

本书是集体劳动的结晶，得到了许多同志的帮助和支持，在此谨向他们表示由衷的感谢！需特别指出的是：在本书的编写和出版过程中，北京希望电脑公司资料部的秦人华经理给予了悉心的指导和大力的帮助；借此机会向她表示深切的谢意和崇高的敬意！

由于成书仓促，书中难免有错误和不当之处，敬请广大同仁不吝指正。

编　者

1991年11月于北京

目　　录

（上　册）

第一部分　Turbo Basic 程序设计概述

第一章　结构化程序设计的概念 …… 2

§1.1　计算机程序设计概述 …… 2

§1.1.1　什么叫程序? …… 2

§1.1.2　机器语言程序 …… 2

§1.1.3　符号程序 …… 2

§1.1.4　程序的种类 …… 3

§1.2　应用程序的开发过程 …… 4

§1.2.1　程序开发过程概述 …… 4

§1.2.2　程序开发步骤详述 …… 5

§1.3　Turbo　Basic 的特性 …… 10

§1.3.1　Basic 是一种通用语言 …… 10

§1.3.2　Turbo　Basic 是 Basic 的一个独特版本 …… 10

§1.4　设计好程序的方法 …… 11

§1.4.1　结构化程序设计 …… 11

§1.4.2　模块化程序设计 …… 11

§1.4.3　自顶向下程序设计 …… 12

§1.5　Turbo　Basic 程序样例 …… 12

§1.5.1　问题的确立 …… 12

§1.5.2　输入格式 …… 12

§1.5.3　输出格式及其确定 …… 13

§1.5.4　程序说明 …… 13

§1.5.5　作为程序规划工具的伪代码概述 …… 22

§1.6　本章辅导 …… 23

§1.6.1　本章小结 …… 23

§1.6.2　本章自测 …… 24

§1.6.3　复习题 …… 25

§1.6.4　编程题 …… 26

第二章　Turbo　Basic 结构化程序设计 …… 28

§2.1　硬件的一般性知识 …… 28

§ 2.1.1 键盘的使用 …… 30
§ 2.1.2 监视器 (Monitor) …… 33
§ 2.1.3 打印机 …… 33
§ 2.1.4 磁盘驱动器 …… 34
§ 2.1.5 微处理器 …… 37
§ 2.2 软件介绍 …… 38
§ 2.3 访问计算机的方法 …… 40
§ 2.3.1 开机 …… 40
§ 2.3.2 装入操作系统 …… 40
§ 2.3.3 计算机系统的重新引导 …… 42
§ 2.4 准备一份 Turbo Basic 软件的工作拷贝 …… 42
§ 2.4.1 软盘系统 (A 和 B 驱动器，没有硬盘) …… 43
§ 2.4.2 硬盘系统 …… 44
§ 2.5 Turbo Basic 程序概述 …… 44
§ 2.5.1 进入 Turbo Basic …… 45
§ 2.5.2 Turbo Basic 屏幕 …… 46
§ 2.5.3 文件命令和编辑命令 …… 51
§ 2.5.4 使用 Turbo Basic 编辑器输入程序 …… 56
§ 2.5.5 键盘上的特殊功能键 …… 61
§ 2.5.6 Run 运行命令 …… 62
§ 2.5.7 Options 命令 …… 63
§ 2.6 本章辅导 …… 69
§ 2.6.1 本章小结 …… 69
§ 2.6.2 本章自测 …… 69
§ 2.6.3 实例 …… 70
§ 2.6.4 复习题 …… 72
§ 2.6.5 调试练习 …… 72
§ 2.6.6 编程题 …… 73
第三章 输入／处理／输出：编写基本程序 …… 75
§ 3.1 数据处理循环：输入／处理／输出 …… 75
§ 3.2 结构化程序设计概念的回顾 …… 76
§ 3.3 数据结构 …… 77
§ 3.3.1 计算机处理的数据类型 …… 77
§ 3.3.2 定义变量名 …… 80
§ 3.3.3 定义常量 …… 82
§ 3.4 INPUT 语句 …… 84
§ 3.4.1 键入数字变量 …… 84
§ 3.4.2 键入字符串变量 …… 85

§ 3.4.3 没有输入任何数据而按 Enter 键会怎样 …… 86
§ 3.4.4 一行中输入多个变量 …… 87
§ 3.4.5 用问号？对输入到字符串变量的值定界 …… 88
§ 3.5 赋值语句 …… 91
§ 3.5.1 使用赋值语句执行基本算术运算 …… 91
§ 3.5.2 用赋值语句给变量赋固定值 …… 104
§ 3.5.3 使用带有字符串变量和字符数据的赋值语句 …… 106
§ 3.6 打印语句 PRINT 和 LPRINT …… 109
§ 3.6.1 一般格式 …… 109
§ 3.6.2 改善输出的可读性 …… 114
§ 3.6.3 打印或显示行太长时的处理方法 …… 116
§ 3.6.4 PRINT 和 LPRINT 语句中分隔符的功能 …… 121
§ 3.7 完善程序 …… 125
§ 3.7.1 注释 …… 125
§ 3.7.2 每次运行前清屏 …… 125
§ 3.7.3 指令续行 …… 125
§ 3.7.4 有关数字变量存贮十进制数的警告 …… 126
§ 3.8 本章辅导 …… 127
§ 3.8.1 本章小结 …… 127
§ 3.8.2 本章自测 …… 128
§ 3.8.3 实例 …… 128
§ 3.8.4 复习题 …… 130
§ 3.8.5 调试练习 …… 132
§ 3.8.6 编程题 …… 133

第二部分　逻辑控制结构细述

第四章　IF 选择语句 …… 136
§ 4.1 四种逻辑控制结构概述 …… 136
§ 4.1.1 顺序结构 …… 136
§ 4.1.2 选择结构 …… 137
§ 4.1.3 迭代结构 …… 137
§ 4.1.4 Case 结构 …… 138
§ 4.2 IF 选择语句块 …… 139
§ 4.2.1 IF 语句块的指令格式 …… 139
§ 4.2.2 比较的规则 …… 149
§ 4.2.3 ASCII 码排序序列 …… 153
§ 4.2.4 条件满足时不操作 …… 154
§ 4.3 IF 语句的其它选择方式 …… 156

§ 4.3.1 IF 语句块与标准 IF 语句的对比 …… 156
§ 4.3.2 条件句嵌套 …… 157
§ 4.3.3 复合条件语句 …… 161
§ 4.3.4 取反条件句 …… 169
§ 4.4 利用行号和跟踪功能简化程序调试 …… 173
§ 4.5 本章辅导 …… 174
§ 4.5.1 本章小结 …… 174
§ 4.5.2 本章自测 …… 176
§ 4.5.3 实例 …… 178
§ 4.5.4 复习题 …… 180
§ 4.5.5 调试练习 …… 181
§ 4.5.6 编程题 …… 181

第五章 关于逻辑控制及自顶而下编程的迭代、循环和子程序 …… 185
§ 5.1 四种逻辑控制结构概述 …… 185
§ 5.1.1 顺序结构 …… 185
§ 5.1.2 选择结构 …… 186
§ 5.1.3 迭代结构 …… 187
§ 5.1.4 分情形结构 …… 188
§ 5.2 迭代方法 …… 189
§ 5.2.1 关于迭代的 DO WHILE … LOOP 结构 …… 189
§ 5.2.2 用于迭代的 DO WHILE … LOOP 结构变形 …… 199
§ 5.2.3 在迭代中使用 EXIT 语句 …… 201
§ 5.2.4 嵌套迭代 …… 202
§ 5.3 FOR … NEXT 语句 …… 207
§ 5.3.1 FOR … NEXT 作为另一种迭代方法 …… 207
§ 5.3.2 FOR … NEXT 的详细说明 …… 211
§ 5.4 迭代方法的选择 …… 220
§ 5.5 在子程序中引入自顶而下模块化编程方法 …… 222
§ 5.5.1 为什么使用子程序 …… 222
§ 5.5.2 GOSUB 的格式 …… 226
§ 5.5.3 嵌套 GOSUB …… 228
§ 5.5.4 程序中 GOSUB 的放置 …… 230
§ 5.5.5 常见的 GOSUB 错误 …… 235
§ 5.5.6 GOSUB 特性总结 …… 237
§ 5.6 本章辅导 …… 239
§ 5.6.1 本章小结 …… 239
§ 5.6.2 本章自测 …… 242
§ 5.6.3 实例 …… 242

§ 5.6.4 复习题 ………………………………………………………… 245
§ 5.6.5 调试练习 ………………………………………………………… 246
§ 5.6.6 编程题 ………………………………………………………… 247

第三部分 打印

第六章 利用输入文件或键入数据显示和打印设计良好的报表 …………… 251
§ 6.1 文件处理导论 ………………………………………………………… 251
§ 6.1.1 什么是文件? ………………………………………………………… 251
§ 6.1.2 文件中的记录 ………………………………………………………… 251
§ 6.1.3 建立顺序文件 ………………………………………………………… 252
§ 6.1.4 从顺序文件读入记录 ………………………………………………… 261
§ 6.2 打印输出的特点 ………………………………………………………… 263
§ 6.2.1 显示与打印输出 ………………………………………………………… 263
§ 6.2.2 打印和显示输出所用的方法 ………………………………………… 264
§ 6.3 编辑功能 ………………………………………………………… 266
§ 6.3.1 概述 ………………………………………………………… 266
§ 6.3.2 LPRINT USING 和 PRINT USING 语句 …………………………… 268
§ 6.4 其它打印选择 ………………………………………………………… 279
§ 6.4.1 使用打印格式表 ………………………………………………………… 279
§ 6.4.2 TAB 函数 ………………………………………………………… 279
§ 6.4.3 Width 语句 ………………………………………………………… 281
§ 6.4.4 SPC 函数 ………………………………………………………… 282
§ 6.4.5 控制一行的打印 ………………………………………………………… 283
§ 6.4.6 进纸至新页 ………………………………………………………… 285
§ 6.4.7 打印页号 ………………………………………………………… 288
§ 6.4.8 打印运行日期 ………………………………………………………… 289
§ 6.5 打印不种类型的报表 ………………………………………………………… 291
§ 6.5.1 明细或事务报表 ………………………………………………………… 291
§ 6.5.2 例外表 ………………………………………………………… 292
§ 6.5.3 一览表 ………………………………………………………… 292
§ 6.5.4 明细表、例外表及一览表的示范打印程序 ……………………… 292
§ 6.6 本章辅导 ………………………………………………………… 294
§ 6.6.1 本章小结 ………………………………………………………… 294
§ 6.6.2 本章自测 ………………………………………………………… 296
§ 6.6.3 实例 ………………………………………………………… 297
§ 6.6.4 复习题 ………………………………………………………… 302
§ 6.6.5 调试练习 ………………………………………………………… 303
§ 6.6.6 编程题 ………………………………………………………… 304

第七章　控制中断过程：自顶向下模块化方法 …… 309
§7.1　控制中断过程的介绍 …… 309
§7.1.1　控制中断过程的例子 …… 309
§7.1.2　单级控制中断 …… 314
§7.1.3　精确表达以提高控制中断报告的质量 …… 323
§7.1.4　单级中断过程小结 …… 328
§7.2　多级控制中断 …… 330
§7.3　本章辅导 …… 340
§7.3.1　本章小结 …… 340
§7.3.2　本章自测 …… 341
§7.3.3　实例 …… 342
§7.3.4　复习题 …… 347
§7.3.5　调试练习 …… 347
§7.3.6　编程题: …… 348

第四部分　程序调试

第八章　调试程序、确认数据和改善程序设计 …… 353
§8.1　程序错误 …… 353
§8.1.1　语法错误 …… 353
§8.1.2　逻辑错误 …… 356
§8.1.3　使用全面且现实的测试数据将减少未发现的逻辑错误 …… 358
§8.1.4　修正逻辑错误 …… 359
§8.2　通过确认输入避免逻辑错误 …… 368
§8.2.1　为什么必须输入确认? …… 368
§8.2.2　无效输入引起的一些后果 …… 369
§8.2.3　数据验证方法 …… 370
§8.2.4　变量的类型及其影响 …… 372
§8.2.5　用 SELECT CASE 语句确认数据 …… 376
§8.2.6　数据确认的其它方法 …… 378
§8.3　输入错误的处理 …… 379
§8.3.1　中止程序的运行 …… 379
§8.3.2　打印一个包含关键变量、出错变量内容和 …… 380
§8.3.3　要么部分处理错误数据，要么整个跳过错误数据 …… 380
§8.3.4　错误数目超过预定的极限时中止运行 …… 380
§8.3.5　用开关“标记”错误 …… 381
§8.3.6　处理输入文件或大量的变量时打印控制总数 …… 382
§8.3.7　发现错误时让用户中止处理 …… 382
§8.4　什么时候应确认数据? …… 384

§ 8.5 用于调试和数据处理的 READ 和 DATA 语句 …… 384
§ 8.5.1 READ 和 DATA 语句的用法：概述 …… 384
§ 8.5.2 什么时候使用 READ 和 DATA 语句？ …… 387
§ 8.6 程序设计特性的回顾 …… 388
§ 8.7 本章辅导 …… 390
§ 8.7.1 本章小结 …… 390
§ 8.7.2 本章自测 …… 391
§ 8.7.3 实例 …… 392
§ 8.7.4 复习题 …… 396
§ 8.7.5 调试练习 …… 397
§ 8.7.6 编程题 …… 398

第九章 深入了解 Turbo Basic 环境 …… 402
§ 9.1 Turbo Basic 环境概述 …… 402
§ 9.1.1 帮助功能 …… 402
§ 9.1.2 选取命令和选项 …… 403
§ 9.1.3 下拉选单和上托选单 …… 404
§ 9.1.4 确定运行状态 …… 405
§ 9.2 重新配置屏幕 …… 406
§ 9.2.1 Tile 选项 …… 406
§ 9.2.2 Zoom 功能 …… 408
§ 9.2.3 Stack 选项 …… 409
§ 9.2.4 改变当前窗口 …… 410
§ 9.2.5 将窗口从屏幕上清除 …… 412
§ 9.2.6 在屏幕上恢复已关闭的窗口 …… 413
§ 9.3 进一步讨论 Window 命令 …… 414
§ 9.3.1 激活窗口 …… 414
§ 9.3.2 Window 命令的 Goto 选项 …… 414
§ 9.3.3 Window 选项小结 …… 414
§ 9.4 Turbo Basic 屏幕底部 Help 行的解释 …… 416
§ 9.4.1 在命令选取之前出现在 Turbo Basic 主屏幕上的提示 …… 416
§ 9.4.2 帮助行随所执行命令的不同而变化 …… 417
§ 9.4.3 热键 …… 418
§ 9.5 改变窗口大小 …… 419
§ 9.6 用不同的配置文件建立不同的环境 …… 422
§ 9.7 Setup 命令下的 Color 选项 …… 423
§ 9.8 Setup 命令下的 Miscellaneous(杂项)选项 …… 424
§ 9.8.1 Auto Save Edit(自动保存编辑内容) …… 424
§ 9.8.2 Backup Source Files(备份源文件) …… 425

§ 9.9 其他的 Turbo Basic 功能 …… 426
§ 9.9.1 File 命令下的 Options 选项 …… 426
§ 9.9.2 进一步介绍 Options 选项 …… 428
§ 9.9.3 Edit 命令详述 …… 431
§ 9.10 本章辅导 …… 436
§ 9.10.1 本章小结 …… 436
§ 9.10.2 本章自测 …… 438
§ 9.10.3 复习题 …… 438
§ 9.10.4 编程题 …… 439

（下 册）

第五部分 Turbo Basic 的高级特性

第十章 数组及表格处理 …… 442
§ 10.1 数组介绍 …… 442
§ 10.2 数组用于存储数字或字符串变量 …… 442
§ 10.2.1 DIM 语句定义数组的大小 …… 443
§ 10.2.2 使用下标访问数组元素 …… 444
§ 10.2.3 Option Base 语句 …… 445
§ 10.2.4 下标可以是整数、变量或算术表达式 …… 446
§ 10.2.5 用 FOR … NEXT 处理数组 …… 446
§ 10.2.6 把数据读入数组 …… 448
§ 10.3 数组和下标的几点补充 …… 450
§ 10.3.1 数组中的元素不一定要连续处理 …… 450
§ 10.3.2 数组中元素能定义一个范围,可以不是 1～n …… 450
§ 10.3.3 由字符串变量组成的数组 …… 451
§ 10.4 使用数组存储总和 …… 452
§ 10.4.1 概述 …… 452
§ 10.4.2 储存总和的数组必须初始化 …… 452
§ 10.4.3 把输入数据累加到数组 …… 453
§ 10.4.4 累加到数组前要有效化输入数据 …… 454
§ 10.4.5 打印数组数据 …… 4554
§ 10.5 数组元素排序 …… 459
§ 10.5.1 替换方法 …… 459
§ 10.5.2 冒泡排序 …… 463
§ 10.6 数组处理的几个补充概念 …… 467
§ 10.6.1 界限选项 …… 467

§ 10.6.2 动态及静态数组 …… 468
§ 10.7 使用数组存储表格数据 …… 471
§ 10.7.1 并行数组 …… 472
§ 10.7.2 把数据存入表格 …… 472
§ 10.7.3 找出与销售输入相匹配的表格条目 …… 473
§ 10.7.4 找不到匹配编码怎么办 …… 476
§ 10.7.5 根据表格中的相对位置寻找表格输入 …… 478
§ 10.8 多维数组 …… 482
§ 10.8.1 二维数组用于存储数据 …… 482
§ 10.8.2 二维数组用于累加总和 …… 487
§ 10.8.3 二维数组用于存储表格 …… 492
§ 10.9 折半查询二维数组 …… 496
§ 10.9.1 串行查询 …… 496
§ 10.9.2 折半查询 …… 497
§ 10.9.3 折半查询的示例 …… 498
§ 10.10 内部数组与外部数组 …… 499
§ 10.11 本章辅导 …… 500
§ 10.11.1 本章小结 …… 500
§ 10.11.2 本章自测 …… 502
§ 10.11.3 实例 …… 502
§ 10.11.4 复习题: …… 506
§ 10.11.5 调试练习 …… 506
§ 10.11.6 编程题: …… 507

第十一章 用过程进行自顶向下模块化程序设计详述 …… 513
§ 11.1 自顶向下模块化程序设计的回顾 …… 513
§ 11.1.1 为什么要编写自顶向下模块化程序 …… 513
§ 11.1.2 自顶向下模块化程序中的 GOSUB …… 514
§ 11.2 在自顶向下模块化程序中使用过程 …… 517
§ 11.2.1 基本格式 …… 517
§ 11.2.2 利用过程传递参数 …… 519
§ 11.3 过程及子程序小结 …… 538
§ 11.3.1 何时使用子程序 …… 538
§ 11.3.2 子程序应是内相关的 …… 538
§ 11.3.3 何时使用子程序，何时使用过程 …… 538
§ 11.4 层次图是自顶向下模块化程序的规划工具 …… 539
§ 11.5 本章辅导 …… 545
§ 11.5.1 本章小节 …… 545
§ 11.5.2 本章自测 …… 547

§ 11.5.3 实例 ……………………………………………… 547
§ 11.5.4 复习题 ……………………………………………… 551
§ 11.5.5 调试练习 ……………………………………………… 553
§ 11.5.6 编程题 ……………………………………………… 554

第十二章 自顶向下模块化程序设计中的函数、字符串处理及递归 ……… 555
§ 12.1 函数用作自顶向下、模块化结构 ……………………………… 555
§ 12.1.1 函数简介 ……………………………………………… 555
§ 12.1.2 自定义函数 ……………………………………………… 556
§ 12.1.3 预定义函数 ……………………………………………… 565
§ 12.2 使用预先定义的函数进行字符串处理 ………………………… 569
§ 12.2.1 输入字符串数据的方法 ……………………………… 569
§ 12.2.2 举例: 如何用字符串函数对字符串进行操作 ……………… 574
§ 12.3 在过程和函数中说明变量 ……………………………………… 582
§ 12.3.1 局部变量 ……………………………………………… 582
§ 12.3.2 共享变量 ……………………………………………… 585
§ 12.3.3 静态变量 ……………………………………………… 587
§ 12.4 递归 ……………………………………………… 589
§ 12.4.1 递归的定义 ……………………………………………… 589
§ 12.4.2 Turbo BASIC 中递归使用举例 ……………………………… 592
§ 12.5 CHAIN (链接) 语句 ……………………………………………… 595
§ 12.6 本章辅导 ……………………………………………… 596
§ 12.6.1 本章小结 ……………………………………………… 596
§ 12.6.2 本章自测 ……………………………………………… 598
§ 12.6.3 实例 ……………………………………………… 600
§ 12.6.4 复习题 ……………………………………………… 601
§ 12.6.5 调试练习 ……………………………………………… 603
§ 12.6.6 编程题 ……………………………………………… 603

第十三章 文件处理和数据库管理 ……………………………………… 605
§ 13.1 主文件处理的系统概述 ……………………………………… 605
§ 13.1.1 什么叫主文件 ……………………………………………… 605
§ 13.1.2 典型的主文件过程 ……………………………………… 606
§ 13.1.3 文件处理程序和软件包 ……………………………… 607
§ 13.2 修改顺序文件 ……………………………………………… 607
§ 13.2.1 顺序文件结构概述 ……………………………………… 607
§ 13.2.2 顺序主文件和细目文件 ……………………………… 608
§ 13.2.3 修改过程 ……………………………………………… 610
§ 13.3 修改过程中的有效性检验 ……………………………………… 624

§ 13.3.1 新账目的检验 ······ 624
§ 13.3.2 双重检验关键字段 ······ 626
§ 13.3.3 关于从顺序主文件中删除编码和删险记录的检验 ······ 67
§ 13.3.4 顺序错误的检验 ······ 628
§ 13.4 为各主记录修改带多个细目记录的过程 ······ 630
§ 13.5 随机文件结构 ······ 634
§ 13.5.1 顺序文件与随机文件中的区别 ······ 634
§ 13.5.2 处理顺序文件和随机文件中的区别 ······ 635
§ 13.5.3 随机文件处理 ······ 636
§ 13.6 二进制文件：简述 ······ 661
§ 13.7 本章辅导 ······ 662
§ 13.7.1 本章小结 ······ 662
§ 13.7.2 本章自测 ······ 664
§ 13.7.3 实例 ······ 664
§ 13.7.4 复习题 ······ 670
§ 13.7.5 调试练习 ······ 670
§ 13.7.6 编程题 ······ 671
第十四章 图形和声音 ······ 673
§ 14.1 Turbo Basic 图形概述 ······ 673
§ 14.2 显示图形所需的微型设备 ······ 675
§ 14.2.1 定义 ······ 675
§ 14.2.2 监视器的类型 ······ 675
§ 14.3 Turbo Basic 图形指令 ······ 677
§ 14.3.1 Screen 指令 ······ 677
§ 14.3.2 馅饼图 ······ 679
§ 14.3.3 直方图 ······ 686
§ 14.3.4 线条图 ······ 692
§ 14.3.5 COLOR 指令 ······ 697
§ 14.3.6 用于画线的 PSET 指令 ······ 698
§ 14.3.7 DRAW 指令 ······ 699
§ 14.4 用 Turbo Basic 产生声音 ······ 701
§ 14.5 结论 ······ 703
§ 14.6 本章辅导 ······ 704
§ 14.6.1 本章小结 ······ 704
§ 14.6.2 本章自测 ······ 706
§ 14.6.3 实例 ······ 706
§ 14.6.4 复习题 ······ 708
§ 14.6.5 调试练习 ······ 709

§ 14.6.6 编程题 ······ 710

习题解答 ······ 712

附录 ······ 794
附录 A Turbo Basic 保留字 ······ 795
附录 B Turbo Basic 字符集 ······ 797
附录 C ASCII(美国信息交换标准代码)表 ······ 798
附录 D Turbo Basic 编辑命令表 ······ 802
附录 E Turbo Basic 错误信息 ······ 812
附录 F Turbo Basic 指令集 ······ 826

第　一　部　分
Turbo Basic 程序设计概述

本 部 分 内 容

第一章　结构化程序设计的概念
第二章　Turbo　Basic 结构化程序设计
第三章　输入／处理／输出：编写基本程序

第一章　结构化程序设计的概念

本章要求:

1. 了解Basic作为普及型程序设计语言的原因，以及Turbo Basic 与其他Basic的区别.
2. 了解结构化程序设计的概念.
3. 了解Basic的演变历史.
4. 了解Turbo Basic 语言的概况.

§1.1　计算机程序设计概述

§1.1.1　什么叫程序?

不管计算机如何复杂，它的动作均由各条计算机指令控制. 这些指令读入那些被称为"输入 (input)"的数据，并将它进行处理，然后生成称作"输出 (Output)"的有意义信息. 将输入转化为输出的指令集合就叫作"程序 (program)". 程序由计算机专业人员，即程序员 (programmer) 编写. 因而，计算机仅能按所编制程序的效能处理数据.

§1.1.2　机器语言程序

计算机执行的所有指令必须为计算机本身的机器语言 (machine language) 形式对于程序员来说，编制这种形式的指令是麻烦和乏味的. 他 (她) 们需要引用内存中的实际地址和存贮单元，使用复杂的指令代码.

§1.1.3　符号程序

由于机器语言的程序设计非常困难，因而程序设计语言进一步开发为可使程序员编写易于阅读和编码的符号指令. 这种语言被称为符号程序设计语言 (symbolic programming Languages). 不过，符号程序在它可以执行或运行之前，必须翻译成计算机本身的机器语言. 计算机自己使用一个翻译程序完成这种转换，使之成为机器语言.

许多符号程序设计语言都可翻译成机器语言. Basic 就是其中之一，而 Borland 公司推出的 Turbo Basic 是该语言的一种新版本. 因为 Basic 相对来说易学，且适用于许多的应用场合，所以它是一种被广泛使用的语言. 其他的符号程序设计语言有: Pasal, C, Ada 和 COBOL 等.

§ 1.1.4　程序的种类

软件（software）是用来描述所有类型程序的术语．大多数软件都是用符号程序设计语言编写的．软件一般分为二大类：

(1) 包括专用程序和程序包在内的应用程序；

(2) 操作系统程序．

下面简要介绍各类软件．

§ 1.1.4.1　应用软件：专用程序和程序包

应用程序指的是那些由程序员提供给用户满足他们需要的程序．有些应用程序是为了快速解决一个一次性的问题而编制的；而其他的应用程序则是为了定期地反复使用而编制的．一个显示一组作为输入数据输入的考试成绩的平均分数的应用程序就是一个一次性问题的例子．而打印学生每学期成绩单的应用程序则是一个可反复地使用的程序实例．

一般来说，应用程序先读取输入，然后将它进行处理，并生成用户需要的信息或输出．

当应用程序为特定的用户而编写时，它就叫作专用程序（Customized program）．Turbo Basic 非常适合作为这类应用程序的编写语言．虽然本文的着重点在于 Turbo Basic 的应用程序设计，但本节也简要描述其他类型的软件．

假如计划由一个程序完成的任务比较规范，如制定预算或创造一个工资计算文件，那么程序包（Program package）可能是用诸如 Turbo Basic 这类语言编写专用应用程序的另一种方法．这样的程序包由软件销售商经营．传统上，由生产厂家以用户手册的形式提供有关如何使用程序包的文档．例如，Lotus 1–2–3 就是一个广泛应用于财政预算，制表和制定试算表等方面的程序包．

如果有一个可用于某种应用场合的程序包，那么买程序包通常比编写专业应用程序更划算，更容易些．不过，若对任一类程序有特殊要求，则编写一个专用程序通常要好些，而不去修改已有的程序包．

§ 1.1.4.2　操作系统软件

在计算机可执行一个应用程序或程序包之前，一组被称为“操作系统（Operating System）”的特殊程序必须装入．这些操作系统程序监控计算机的总体操作．也就是说，操作系统监视或控制应用程序或程序包所完成的所有功能．

IBM 微型计算机（及其兼容机）上用得最多的操作系统为 DOS 和 OS／2．DOS 为 Disk Operating System（磁盘操作系统）的缩写．MS–DOS 是 DOS 的一个通用性版本，而 PC–DOS 为 IBM 指定的 DOS 版本；OS／2 为 Operating System／2（操作系统／2）的缩写，可用来同时运行多个应用程序．下章将进一步介绍操作系统．

§ 1.2　应用程序的开发过程

§ 1.2.1　程序开发过程概述

许多人认为计算机程序设计始于编写程序指令，并以程序的测试而告终。然而，事先毫无准备的程序员经常编写出设计差或不合适的程序。

程序设计所包含的步骤应改进，编码只有在程序需求全部指定完毕和逻辑关系清楚确定后，才能进行。

另外，如果它要用于正式的生产过程，那么在程序编码和测试完成之后还有步骤。各个反复使用的程序必须归档。这就意味着，程序必须带有一套介绍其用法的正式步骤和说明。这个资料（documentation）供如下人员使用：

(1)　输入数据和／或使用输出的用户；

(2)　保持程序工作的软件维护人员。

程序开发过程中包含的步骤概述如下。其中各步骤将随后详细讨论。

程序开发过程

1. 获取程序指标

   程序员经常为特定的部门或公司中的用户编写程序。在大公司或需要长且复杂程序的公司中，程序员可能要处于负责设计总体商业系统和制定对各应用程序需求的系统分析员的管理之下。在小公司中，程序员可能同时负责系统设计和要求的程序。无论有否系统分析员监督工作，程序员均应确信了解了用户的精度要求。

2. 用规划工具规划程序

   实际编写程序之前，程序员使用诸如伪代码（pseudo-code）和层次图（hierarchy charts）帮助画出程序的逻辑结构图。本文自始至终将介绍这些规划工具的用法。

3. 编码和输入程序

   在程序被规划完毕之后，程序员便写出程序，并将它输入计算机系统之中。编写程序的另一个叫法为“编码”。在大多数情况下，程序是用键盘敲入，并在屏幕上显示的。

4. 将程序翻译成机器语言

   如前所述，符号程序在它运行之前必须翻译。翻译程序翻译时，检查规则违反情况或语法错误。如果有这种错误，翻译程序则将其列出，因而该程序需进行修正，然后在它运行之前重新翻译。

5. 程序测试。

   程序必须用测试数据运行或执行，以保证读取输入.处理和生成输

出等的正确性。即，程序测试为的是消除任何导致错误输出的逻辑错误。若有逻辑错误，程序将要进行修正、翻译、然后再次测试。

6. 编写程序文档

为便于用户能够运行程序并清楚其实现的功能，程序员应编写一本手册或一套说明.

---

§1.2.2 程序开发步骤详述

§1.2.2.1 获取程序指标

如前所述，程序读取输入，并将它进行处理，然后再产生输出．因此，输入，处理和输出的需求必须在程序规划前指定．程序员应与用户讨论如下问题:

a. 输入的确切形式．例如，输入哪种数据类型，其格式如何，等等．

b. 进行处理的确切形式．例如，如何处理有效数据，如何处理无效数据，以及将进行哪种运算等。

c. 输出的确切形式．例如，标明输出的标题是什么，输出是在屏幕上显示还是打印出来，应打印的内容等．

§1.2.2.2 用规划工具规划程序

程序员在开始编码之前应先规划一下要用于程序中的逻辑关系．就像建筑师在建造一座建筑物之前先画蓝图一样，程序员应在编码之前先对程序进行规划．这样将使程序要使用的逻辑结构定型，并使随后出现的错误减少到最小程度。

程序员最常用的两种规划工具为:

a. 描述程序中逻辑结构的伪代码.

b. 描述程序中模块或段间关系的层次图

伪代码采用与英语类似表达式或句子描述将用于程序中的逻辑结构．虽然流程图也时常用作规划工具，但伪代码一般来说更好；这是因为它描述大型、复杂程序的设计时结构清晰，简便．

假如要输入一组数据，并在输入完毕后打印总和，那么在编程实现它之前，先用如下伪代码规划:

```
START
    输入一个数据
    DO  WHILE  还有数据要处理
        将数据加入总和
        输入另一个数据
```

```
    ENDDO
    打印总和
STOP
```

由于介绍的程序比较简单，因而读者刚开始觉得这个工具没有必要。然而，以后设计带有复杂逻辑结构的中、高级程序时，伪代码将证明为非常有价值的工具。本章后面部分将简要介绍伪代码，并在第三章至第五章中将它作为规划工具详细介绍。

层次图用于规划被细分为称作"模块 (module)"的段的复杂程序。第七、十一和十二章将详细介绍层次图。

§1.2.2.3 编码和输入程序

程序员用符号程序设计语言编写一系列指令，即所谓的"源程序 (Source Program)"。在它被翻译成机器语言之前，源程序不能执行或运行。

源程序先用键盘键入计算机，然后存放在存储介质，通常为磁盘之上。

§1.2.2.4 将程序翻译成机器语言

源程序被输入后，计算机将它翻译成机器程序，即目标程序 (Object Program)。象 Basic 这样的高级符号程序设计语言有两种翻译程序：解释器和编译器。

解释器 (interpreter) 是一种单步逐行扫描并执行源代码的翻译程序。对于程序员来说，单步之中翻译和执行一起进行具有一定的灵活性。不过，因为解释器必须一遍又一遍地扫描相同程序行，以保证它与其他行的一致性，因而解释器与其他类型的翻译程序相比较慢。

编译器 (compiler) 是一种先将源程序转换为机器语言程序或目标程序的翻译程序。目标程序在单独的过程中再执行。虽然这分成了两步，但它比解释型程序执行起来更快，效率更高。

由编译生成的目标程序可在任何时候保存和执行，而不必翻译。反之，解释型程序在每次运行之前却必须先翻译。这使得编译程序对于定期性的生产过程更加有效。

大多数 Basic 版本使用解释器，这样虽然使程序员易于输入和测试程序，但导致非常慢的执行速度。用诸如 Pascal，COBOL 和 Ada 之类比较复杂语言编写的程序一般均为编译型，这就意味着程序被非常有效地翻译。虽然 Turbo Basic 是一种编译型 Basic 语言，但它包含许多其他解释型 Basic 所拥有的特性。因而，它既易于使用又可生成高效的机器语言码。

§1.2.2.5 程序测试

a. 调试阶段

计算机翻译一个程序时，编译器或解释器检测到的任何错误将被列出。即，任何程序

设计规划的违反均被标为"Syntax error（语法错误）"。例如，如果将 Turbo Basic 输入一个数据的指令写成 INPUTT 而不是 INPUT（正确形式为 INPUT），计算机将打印一条表明发现语法错误的信息。对于编译型程序而言，语法错误必须修正，并且在程序开始执行之前重新编译。

值得注意的是，翻译过程中检测到的语法错误仅仅是程序设计错误中的一种。可能发生的其他类型错误称作"逻辑错误（logic error）。如果程序设计步骤的顺序安排不当或错误的指令被编写，那么，这些错误将发生。例如，若对两个数进行相加而不是相乘，就将导致一个翻译器不能检测的逻辑错误。

对程序进行测试时，逻辑错误由程员检查。为检测逻辑错误，用测试数据运行程序，观察它是否正确地处理数据。运行测试先将测试数据作为输入，然后对它进行处理，并产生输出。程序员检查输出的正确性。若不正确，便发生了逻辑错误。此时，程序员必须找出错误，将它修正，然后重新翻译该程序，并再次测试。这个过程反复进行，直至程序工作正常为止。消除所有的语法和逻辑错误的过程叫作"调试（debugging）。

测试数据必须认真准备，以确保程序中的所有状态均真正被测。若不然，起初看似正确的程序可能在运行更完备的数据或反复使用时，最终产生逻辑错误。

程序被翻译后，目标程序可立即用测试数据执行，或者存入磁盘以备日后使用。如果程序已经被编译过，保存的目标程序可能在任一时候装入计算机，并执行或运行，而不必重新编译源程序。图 1.1(见下页)描述了编写和测试程序所包含的步骤。

b. 调试方法

据我们所知，一个程序规划和编写完毕后，必须翻译并用测试数据执行。在翻译或执行过程中发生错误不是什么稀罕事。下面给出程序员检查程序错误的几种调试方法：

(1) 桌面检查

程序员应当在程序被键入前进行认真的复查；并在键入后再次检查，桌面检查（Desk checking）将使语法错误和执行或运行时间（run time）错减少，一般来说还可减少调试程序所花的总时间。经验不足的程序员常常忽视桌面检查的必要性，设想计算机能更好地找出错误。然而，取消桌面检查阶段可导致未检测的逻辑错误，这些错误以后可能要花数小时，甚至数天才能找出。经验丰富的程序员在编码完毕后认真复查程序，并在程序键入后再次复查。

(2) 修正语法错误

当程序被翻译时，计算机将打印出指向任何违反规则或语法错误的诊断信息。然后，程序员必须修正这些错误，并在程序可运行测试数据之前重新翻译。正如我们将看到的一样，Turbo Basic 编译器不仅打印错误信息，而且标出不正确的指令，并时常标明指令中不正确的单词，使得上述过程比较容易。

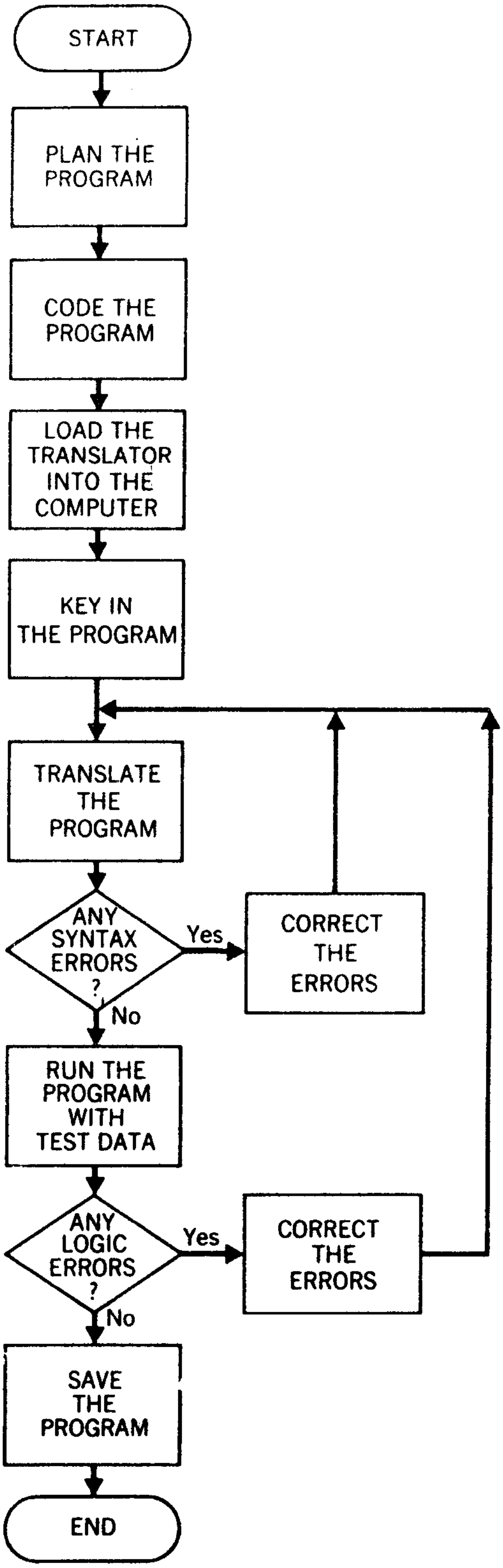


图 1.1　编写和测试程序的步骤

(3) 进行结构化程序的走查

在源程序由计算机以源程序清单形式列出后，程序员通过运行测试数据测试程序的逻辑结构．不过，最好是先走一遍程序，看是否能产生预期的结果．这就是所谓结构化程序的"走查（walkturough)"．在这个过程中，程序员通过人工处理测试数据，看能否获得正确的结果，走一遍程序的逻辑路径．走查在机器执行之前进行，以便帮助程序员甚至在程序运行之前就查出逻辑错误．

对于很长或很复杂的程序，结构走查在程序的实际编码之前，直接在伪代码上进行．这个步骤也可减少以后程序的修改．

(4) 通过执行程序检查逻辑错误．

在许多情况下，通过执行程序检查逻辑错误是调试过程中最困难和最费时间的．如果进行过桌面检查和结构化程序走查，那么可能发生于程序执行过程中的逻辑错误数目将减少到最小程度．第八章重点介绍查找和修正 Turbo Basic 程序中逻辑错误的其他方法．这些方法中的大多数同样适用于其他语言

(5) 用认真准备的测试数据测试程序

如前所述，测试数据的准备是调试过程中一个极其重要的方面．程序员准备的数据应能测试正常操作条件下程序的每一个可能状态．充分测试并运行了一段时间的程序突然发生问题，是很少见的．这些问题的出现常常是因为以前未碰到的特殊状态出现了，以及程序员未对程序充分测试．

§1.2.2.6 编写程序文档

程序开发过程的最后一个组成部分是准备文档，以便用户和计算机工作人员不需向程序员咨询便能运行该程序．文档应提供有关程序如何运行、程序实现的功能、可能发生的错误状态类型等方面的说明．如果文档编写完毕，那么不用原程序员的介入，程序应能维护，甚至修改．

自我测验(一)

自我测验题贯穿全文，并附有解答．其目的在于检查读者是否理解讲述的内容．

1. ________语言的程序才能被计算机执行或运行．
2. 应用程序一般用________语言编写，而不用机器语言；这是因为________．
3. 用机器语言以外的语言编写的程序在执行之前必须________．
4. 用诸如Turbo Basic之类程序设计语言编写的程序叫________程序．
5. Turbo Basic是一种使用________作为翻译程序的Basic语言。这种翻译程序生成

________.

6. ________将源程序转换为________程序.
7. 在程序翻译过程中查出错误通常叫做________错误. 这类错误是由于________.
8. 用测试数据运行一个程序之前, 程序的逻辑结构由________检查.
9. 用编译器翻译完程序后, 它可能________或________.
10. 程序被充分测试后, 它应________, 以便用户不必向程序员咨询便能运行它.

§ 1.3 Turbo Basic 的特性

§ 1.3.1 Basic 是一种通用语言

Basic 是 Beginner's All-Purpose Symbolic Instruction Code (初学者通用符号指令代码) 的缩略词. 不过, 有许多人认为是先提出了 Basic 这个名词, 然后才有该缩略词. Basic 由 Dartmouth College 的 John G. Kemeny 和 Thomas E. Kurtz 于六十年代初期创立的. 它是当时出现的少数几个符号程序设计语言中的一个. 因为 Basic 简单易学, 并且功能强, 灵活性好, 因而立即得到推广, 并广为传播.

今天, 有不下百种的 Basic 版本 (解释型和编译型) 可供大型计算机和微型计算机使用. 事实上, 当微型计算机于七十年代引入时, Basic 是最早的、在大多情况下也是唯一可供使用的符号程序设计语言. 目前, 它仍然是用于微型计算机上最普遍的符号语言, 并一直广泛用于大型计算机.

§ 1.3.2 Turbo Basic 是 Basic 的一个独特版本

虽然有许许多多 Basic 版本可用于微型计算机, 但目前主要的 Basic 版本有: IBM 公司 Advanced Basic (一般称作 BASICA), Microsoft 公司的 GW BASIC 和 Qyucj /basuc, 以及 Borland 公司的 Turbo Basic. 本书着重介绍 Turbo Basic. 不过应注意, 虽然 Turbo Basic 具有一些非常有用和独特的特性, 但其语法与大多数版本的 Basic 语言非常类似.

Turbo Basic 较之 BASICA, GW BASIC 和大多数其他版本 Basic 的 5 个主要优点是: Turbo Basic 是编译型语言, 而不是解释型语言. 这个优点使之成为一种更有效的语言. 大多数用 Turbo Basic 编写的程序比用 BASICA 或 GW BASIC 编写的等效程序执行速度要快四至十倍. 同样, Turbo Basic 编译器每分钟最多可译码 12000 行. 另外, Turbo Basic 与 BASICA 和 GW BASIC 完全兼容; 这就意味着, 用这些解释型版本 Basic 编写的程序可以用 Turbo Basic 编译并执行.

Turbo Basic 的另一个优良特性是它的交互、多窗口显示系统. 屏幕上同时显示四个窗口 (window). 其中一个显示程序, 一个显示输入和输出数据, 一个显示调试帮助信息, 最后一个显示给用户的信息. 虽然大多数其他版本的 Basic 允许查看程序或输入和输出, 但不可同时进行. 然而, Turbo Basic 的四个窗口给程序员提供了良好交互显示.

此外，Turbo Basic 程序员还可根据自己的需要设计或配置这四个窗口．即，每个窗口可以设置成程序员喜爱的大小和颜色（若有彩色显示器的话）．第九章进一步介绍如何修改 Turbo Basic 环境．

最后，Turbo Basic 具有编写设计良好的结构化程序所必需的逻辑控制结构．下面讨论结构化程序的特性．

§1.4 设计好程序的方法

§1.4.1 结构化程序设计

§1.4.1.1 概述

当程序设计于六十年代和七十年代成为一种重要的职业时，程序员的主要任务是使程序正常运转．虽然这是程序员的主要任务，但编写程序要易于阅读，调试，修改和维护现在也被看作一个非常重要的目标．这就是说，随着计算机领域的不断发展，越来越多的注意力集中于程序设计，就如保证程序运行一样．

用任何语言编写设计良好程序的最重要方法是结构化程序设计（structed programming)．结构化程序设计是一种标准化的程序设计方法；无论采用哪种语言编程，它均可使所有程序具有相似的形式．一般来说，结构化程序比非结构化程序易读，也易于调试．如果日后需要修动的话，结构化程序也更容易修改．此外，它还易于评估，以便程序设计负责人和系统分析员能更好地评价程序员的技能，以及鉴定程序中的设计要素．

§1.4.1.2 结构化程序设计是无转向程序设计

对于那些有过程序设计经历的人来说，他们可能接触过非结构化方法，这种方法频繁使用 GOTO（转向）语句（GOTO 是一个描述分支的术语)．GOTO 语句经常使程序逻辑的跟踪变得非常困难．同样，它加大了程序调试的难度．

结构化程序设计的一个主要目的是：通过减少程序中指向“GOTO”语句的入口数目和出口数目，从而简化调试工作．由于这个原因，结构化程序设计常被称作无转向程序设计．

结构化程序设计方法的使用可完全避开 GOTO 语句．在 Turbo Basic 中，程序的执行顺序由 DO WHILE 指令或其他一些逻辑控制语句控制．

§1.4.2 模块化程序设计

长又复杂的程序常被细分为模块（module)，也叫子程序（subroutine)、辅程序(subprogram）或过程（procedure)．模块是完成特定功能的指令序列．被细分成模块的

程序叫模块化程序．在模块化程序中，主程序或模块按需要调用其他模块．这就是说，子程序、辅程序或过程被编成独立的程序段，并在模块化程序的主程序需要时调人。

程序模块不仅单独编写，而且分别测试．这就有可能将大而复杂的程序分成各个程序段．如有必要，不同的程序员可编写和调试不同的程序段．总之，模块化程序由各个程序段所组成；各程序段在主模块的控制下运行．

§ 1.4.3 自顶向下程序设计

另一个重要的、使程序易于阅读、更有效的设计方法叫做“自顶向下程序设计(top-down programming)”．自顶向下程序设计意即正确的程序设计最好在小模块之前设计和编写主模块．因此，在自顶向下程序中，首先编写主程序，然后编写中间程序，最后编写小程序．(本文交替使用模块和程序两术语)．

用这种自顶向下的方法编写模块时，主要注意力集中在程序的组织或流程上．首先编写主模块，其中包括其他计划调用小模块的引用．详细的指令留给最后编写的小模块．

自顶向下程序设计类似于词汇表的编写，先确定框架，待主体结构确定后再给出详细内容．为实现高效的程序设计，这种标准化自顶向下方法是对结构化程序设计的极好补充．

本文中所有程序均采用结构化方法，并彻底避免 GOTO 语句的使用．另外，在比较高级的程序中采用了自顶向下的方法，以便读者学会使用公认的、标准化的有效方法设计程序．

§ 1.5 Turbo Basic 程序样例

§ 1.5.1 问题的确立

一个大公司的计算中心分配到计算全体无工资职员每周的酬金或应得工资．各个雇员的姓名、每小时工资和工作的小时数作为输入提供．每周的工资额按下面公式计算：

每周工资额 ＝ 工作小时数×每小时工资

在开始处理之前，输入数据必须变成计算机可以“阅读”的形式．输入数据可能由键盘键入，可能从磁盘或其他存储介质上读取．

§ 1.5.2 输入格式

假设雇员数据用键盘键入．正如后面将看到的一样，用于键入输入数据的设备并不真正影响程序的逻辑．

各个职员的数据由三个数据项，即变量（Variable）所组成．该三个变量分别命名为：EmpName$，Hours 和 Rate．后面，讨论变量名命名规划时，读者将明白

EmpName $ 中的“$”是不可少的；这是因为它被定义为容纳字符数据的变量.

§ 1.5.3 输出格式及其确定

对于在键盘上键入的各个 EmpName $ ，Hours 和 Rate，我们将在打印机上打印雇员的姓名和他（她）的每周酬金．每周酬金为 Hours 和 Rate 之积.

如前所述，对应于各个雇员的三个变量——EmpName $，Hours 和 Rate——将作为输入读入程序．输出变量 Wages 等于 Hours 和 Rate 之积．每个雇员的姓名将随计算出的 Wages 一同打印。

§ 1.5.4 程序说明

§ 1.5.4.1 复查设计指标

回忆一下，程序开发过程始于程序设计指标的复查．本例的程序设计指标包括输入和输出数据的格式，以及所进行处理的描述．它们已在 1.5.1 节至 1.5.4 节介绍过.

这些设计指标一旦提供，程序员便应规划程序的设计，即画出用于该程序中的逻辑结构．下面给出勾划用于本样例程序中逻辑结构的伪代码．编写伪代码的方法在本章后面介绍．在这里引入这种规划工具是因为：我们认为，甚至在准备自己的伪代码之前，读者便应学会阅读伪代码．下一节给出编写伪代码的基本规划.

```
START
    为用户打印一条描述输入雇员数据的信息
    输入 Empnam $，Hours 和 Rate
    DO  WHILE  还有输入数据
        计算 Wages= Hours×Rate
        打印结果
        输入 EmpName $，Houus 和 Rate
  ENDDO
  打印一条报表结束信息
STOP
```

程序规划好后，程序员准备编程．回忆一下，程序是一组为产生输出而操作输入的指令．下面是一个简化后的 Turbo Basic 程序；它输入用键盘键入的雇员数据，并创建一个包含各位雇员姓名及其工资的工资表.

```
10  'This Program Calculates Weekly Wages
20  PRINT "Enter Name, Hours and Rate
30  INPUT EmpName $, Hours, Rate
```

```
40  DO  WHILE  Hours  <  999
50        LET Wages  =  Hours * Rate
60           LPRINT "Weekly Wages For"; EmpName$;"  is  "; Wages
70           INPUT EmpName$,  Hours,  Rate
80  LOOP
90  LPRINT "End of Report"
100  END
```

§1.5.4.2 编码指南

Turbo Basic 程序中的各行可能有一个行号，但它是不是必须的．这里使用行号是为了便于明确地查阅各行．编行号时用 10 的倍数（10，20，等）以便需要插入时易于插入．例如，如果要在 20 行和 30 行之间增加一行，则可将它的行号编为 25．不过，注意：使用 Turbo Basic 时，指令按顺序执行，而不管其行号如何．如果偶然编成下面形式：

```
10
30
20
```

那么指令将按列出的顺序执行，而不是按行号顺序．

作为一种风格，本书将 Turbo Basic 的关键词，如 PRINT，INPUT 和 DO WHILE，均采用大写；变量混合使用大、小写字符，如 EmpName$，Hours 和 Rate．这个约定用于本书中的程序，但不是必须的．Turbo Basic 是一种大小写不敏感的语言，即编译器忽视大小写字符间的区分．例如：PRINT，print，Print 或 PRint 均被翻成同一个词．

§1.5.4.3 指令

除非诸如 DO WHILE 之类的逻辑控制结构被执行，否则 Turbo Basic 程序中的各条指令按顺序执行．

下面依次讨论各条指令．第一个程序行为：

```
10  'This Program Calculates Weekly Wages
```

该行的行号为 10，是一个注释行（comment line）．它仅仅是为了提供信息的目的而编写的，既不编译，也不执行．注释行用一个撤号或单引号(') 作开始．一个撇号一旦出

现在任一行上，该行的其余部分被当做一条注释，并且不编译或执行。如前所述，任何指令行均可能仅含有一条注释，或者有一条附加注释。例如，下面的 END 指令包含一条不被编译或执行的注释。

```
100 END 'This is the last line of a program
```

因为注释有助于其他人理解程序的编写和逻辑，故建议大量使用注释。在日后测试程序时，注释甚至可帮助程序员想起一些细节。

程序中的第二行为:

```
20 PRINT "Enter Name, Hours and Rate"
```

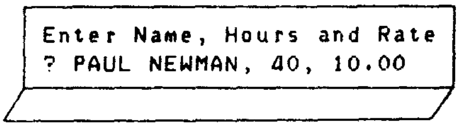
执行时，该行将在屏幕上给用户显示一条信息。该条信息用于帮助用户确定要输入的数据及其形式。在 PRINT 语句中，引号中的字符串在屏幕上原样显示。这些字符串出现在显示输入、输出的 RUN 窗口中。

该 PRINT 指令中的信息叫做字符或字符串常数 (literal or string constant)。不要混淆 PRINT 指令行和注释行。程序中的注释行仅仅是提供给程序员的备注。PRINT 指令在程序运行时给用户打印一条信息。

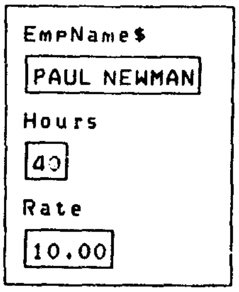
```
30  INPUT EmpName$, Hours, Rate
```

该 INPUT 指令接收用户在键盘上输入的三个数值。执行 INPUT 指令时，计算机在屏幕止打印一个"?"。因为它提示用户计算机等待输入数据，因而"?"被称作提示符 (prompt)。在这种情况下，要输入三个数值—EmpName $，Hours 和 Rate。这三个数据必须由用户键入，并各自用逗号分隔。三个数值输入完毕后，用户必须按 Enter (回车) 键将数据传送到计算机，然后依次执行下一条执行。注意: Enter 键有时用符号"↵"标识，或象打字机上一样，称作 Return (返回键)。

20 行在屏幕上显示一条信息; 30 行获取输入数据。下面是一个可能输入的数据，以及 PRINT 和 INPUT 指令执行时，它们在计算机内存贮情况的例子。



Central Processing
Unit (CPU)



←When the user presses the Enter key, data will be stored in the variables called EmpName$, Hours, and Rate

下面的五条指令被看作一个单元:

```
40      DO  WHILE Hours < 999
           :
80      LOOP
```

DO WHILE … LOOP 叫做逻辑控制结构（logical control construct）. 它指示计算机执行紧跟在 DO WHILE 后面直至 80 行中 LOOP 指令为止的指令序列，并且连续执行这些指令直到 DO WHILE 指令行中指定的条件满足为止. 本例中 DO WHILE 检查 Hours 是否小于（<）999. 这意味着，除非输入的 Hours 大于或等于 999 的数，否则 DO WHILE … LOOP 间的指令将被反复执行. 换句话说，每当输入的 Hours 值小于 999 时，DO WHILE … LOOP 间的指令将被执行.

总之，该程序用一条描述将进行的处理的注释作为开始，然后给用户显示一条信息，并输入第一个雇员的一组数据；接着，DO WHILE … LOOP 语句组处理第一组数据，打印结果，并读取下一个雇员的数据. DO WHILE … LOOP 结构反复执行，直至三个输入数据中为 Hours 准备的值大小或等于 999 为止. DO WHILE … LOOP 结构中指令的缩进仅仅是为了强调它们处于一个逻辑结构的控制之下.

Hours 变量中的数值 999 叫做"终止值（trailer Value）". 它只在所有有效数值均输入完毕后才输入. 为 Hours 输入的输入值 999 表示数据输入结束. 由于 999 不可能是一个实际有效值，所以才将它选作终止值. 这就是说，因为任何雇员在任一指定的星期内不可能实际工作 999 小时以上，故 999 被选来标明运行结束状态.

因此，DO WHILE … LOOP 结构中的指令将反复执行，直至运行结束或终止值999 输入到变量 Hours 中为止。假如用这个程序处理十五名雇员的报表。为结束运行，我们可能输入 STOP、999，999 作为最后一个输入行。因为 Hours 等于 999，所以 DO WHILE … LOOP 将中止运行。注意：虽然只有 Hours 被 DO WHILE 测试，但执行 INPUT 语句时，三个输入条目——EmpName $，Hours，Rate——全要输入。不过，是作为变量 Hours 值的 999 中止这个循环。

到目前为止，我们已经概述一个结构化程序的逻辑结构。该程序始于注释，然后给用户打印一条解释输入数据形式的信息；接着，一个初始 INPUT 语句读取第一组的三个数值；除非输入的数据中有一个终止或运行结束值 999（或更大的值）输入到变量 Hours 中，否则 DO WHILE … LOOP 将处理这些数据，并读取下一组输入值。程序的一般结构将按下面这种结构化的格式：

```
INPUT  第一组数值
DO  WHILE 一个变量不等于终止值
    处理数据
    INPUT  下一组数据
LOOP
  ⋮
```

除非 DO WHILE … END DO 为伪代码结构，否则用于大多数程序的伪代码规划工具将有相同的形式。所有其他术语，如输入、读取，不需要任何的特殊规则：

```
输入数据
DO  WHILE  还有输入数据
          ⋮
      输入数据
END DO
```

这个 Turbo Basic 程序中 DO WHILE … LOOP 结构内的第一条指令处理第一组数据变量：

```
50      LET Wages = Hours * Rate
```

50 行计算工作小时数与每小时金额之积，并送入 Wages。其中“*”是乘号。

下面让我们“走查”该程序的逻辑，看一看数据如何处理。当 INPUT 语句被执行时，如前所述，三个变量的内容被读入。

考虑如下方面：

(1) 出现在 Edit（编辑）窗口内程序中的指令；

(2) 程序被执行时，在 RUN（运行）窗口中输入的数据.

Instruction—Edit Window	Displayed Input—Run Window
`30   INPUT EmpName$, Hours, Rate`	`? LINDA EVANS, 40, 8.00`

计算机将按如下形式将输入数据存入三个变量:

CPU

```
EmpName$
[LINDA EVANS]
Hours
[40]
Rate
[8.00]
```

50 行中的 LET 指令将 40X8.00, 即 320 赋给 Wages:

CPU

```
Wages
[320]
```

←The variable Wages stores the result of 320

60 行是一个输出操作，为用户在打印机上将信息和数据混合打印:

```
60     LPRINT "Weekly Wages For "; EmpName$;" is "; Wages
```

每条 LPRINT 语句在打印机上打印一行. 尽管 20 行中的 PRINT 语句在屏幕上显示数据，而 LPRINT 语句在打印机上打印数据. LPRINT 可将常数或信息随变量内容一起打印. 常数或信息用引号引起来. 在此，LPRINT"Weekly Wages For"将打印引号内的词语. LPRINT EmpName $ 打印变量 EmpName $ 中的内容. 各打印元素由分号分隔.

上面的 LPRINT 指令将在一张纸上打印如下内容:

Contents of EmpName$ Contents of Wages

Printed output

```
Weekly Wages For LINDA EVANS is 320
```

70 行输入下一组雇员姓名,工作小时数和每小时工资:

```
70      INPUT EmpName$, Hours, Rate
```

回忆一下,由用户输入的 EmpName $, Hours 和 Rate 值用逗号分隔. 按 Enter 键后，这三个数值将送入计算机.

80 行标明 DO WHILE 结构的结束.

```
80  LOOP
```

遇到 LOOP 语句时，如果指定的条件（Hours<999）仍成立，则用 DO WHILE 作为开始的指令序列被重复执行. 这就是说，执行 70 行中指令时输入下一组输入数据；而且，如果在 70 行输入的 Hours 小于 999，则 DO WHILE 将重复执行.

总之，LOOP 将控制权回送到 DO WHULE 语句，即如果随后的 Hours<999 则处理下一组输入数据. 假设第二个雇员的 Hours 小于 999，那么将处理第二组输入数据，并打印第二行. 然后，读入第三组输入数据. 接着，计算并打印 Wages、读取新输入数据的过程重复进行. 直至用户输入 Hours 大于或等于 999 为止. 在后面那种情况下，DO WHILE … LOOP 将中断，而 LOOP 后面的语句被执行. LOOP 后面的语句是一条 LPRINT 指令:

```
90  LPRINT "End of Report"
```

这条语句将字符常数"End of Report"在所有的周薪打印之后作为最后一个打印行打

印。

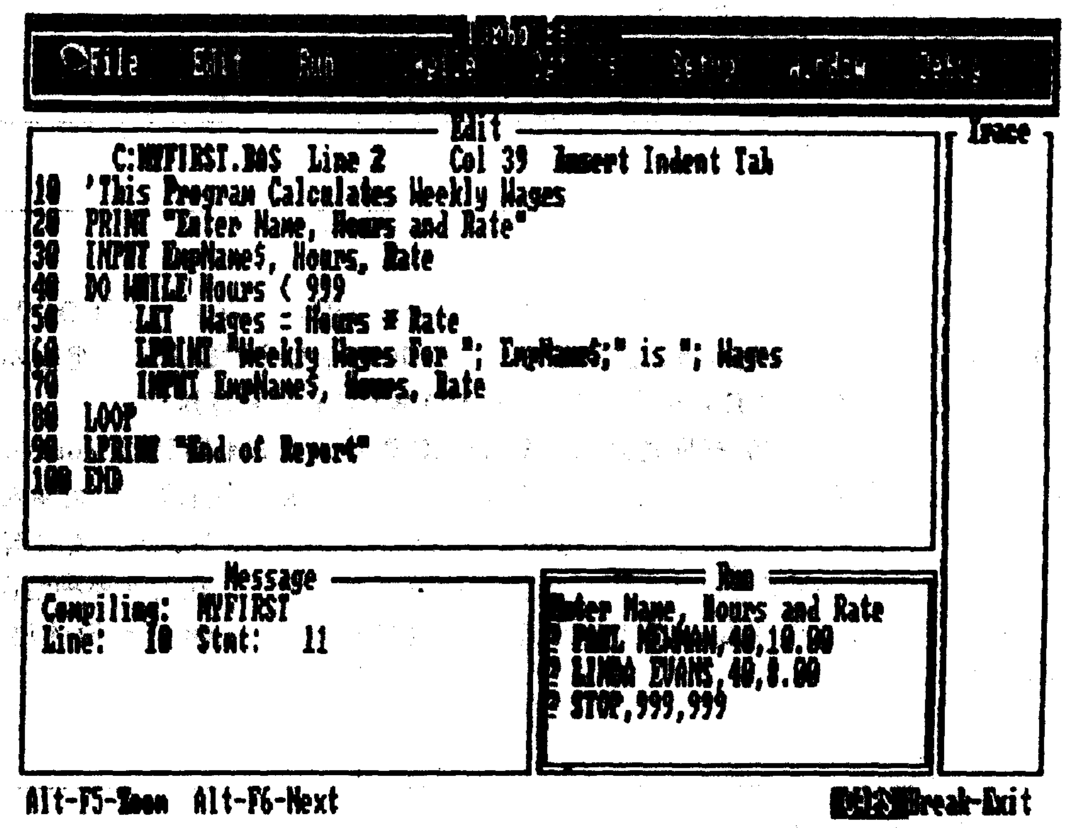
这条 LPRINT 指令只有在 DO WHILE … LOOP 被中止之后才被执行。也就是说，当大于或等于 999 的 Hours 值随 EmpName$ 和 Rate 一起输入时，最后的 LPRINT 语句才被执行。

Turbo Basic 程序中的最后一条指令是 END 指令：

```
100   END
```

虽然这条指令是不必需的,但本书中的所有程序加入这条指令为的是明确程序的结束。

图 1.2 给出上述 Turbo Basic 程序与输入数据样例一起出现在屏幕上的情形。可以看出，程序本身出现在一个标题为 Edit 的盒子，即所谓的编辑窗口（Edit Window）中；提供给用户的打印信息与输入样例一起出现在标题为 RUN 的独立盒子或窗口，即运行窗口（RUN Window）中（下一章介绍其他两个窗口）；打印在纸上的输出也在图中给出。



(a)

图 1.2(a) 出现在屏幕上的 Turbo Basic 程序样例

Printed Output

```
Weekly Wages For PAUL NEWMAN is  400
Weekly Wages For LINDA EVANS is  320
End of Report
```

(*b*)

图 1.2(b) 样例程序输入结果的硬拷贝

§ 1.5.4.4 Turbo Basic 程序要素的回顾

上述指令与所有的 Turbo Basic 指令一样，使用如下要素：

a. 行号:可有可无.

b. 注释：它提供有关程序的信息．注释以一个撇号或单引号作为开始，并可能单独占一行或在任何指令行的末尾.

c. 保留字：它们对编译器有特殊含义，并且常用于执行操作（如，PRINT，INPUT，DO WHILE，LET，LPRINT，LOOP，END）．本书约定将保留字大写.

d. 变量：它是存贮器中用来保存数据的区域（例如，EmpName$，Hours，Rate，Wages）．变量名可混合使用大小写；也可在变量名末尾使用一个"$"，用以保存如下两种数据：(1) 字母数据；(2) 包含字母、数字和特殊字符的数据；这种数据被称为字母数字数据（alphanumeric data）.

e. 算术操作符：

+ （加）
- （减）
* （乘）
/ （除）
^ （乘方）

f. 常数：程序中固定不变的数据（如，40行中的终止值999和90行中的文字"End of Report".

g. 关系运算符：用于DO WHILE … LOOP和条件测试中的关系测试：

<	小于	< =	小于或等于
>	大于	> =	大于或等于
=	等于	< >	不等于

§1.5.5 作为程序规划工具的伪代码概述

一个程序，如上面介绍的，应首先用伪代码规划，设计出逻辑结构。这个规划工具在帮助程序员确定程序的最佳结构时特别有用。此外，对于复杂程序而言，为确保程序被正确设计，这个工具是必不可少的。

伪代码的基本描述规则

1. 伪代码用START作为开始，STOP作为结束。
2. 伪代码中的所有指令顺序阅读。
3. 没有为指定伪代码中的指令而设立规则。即，输入数据、读取数据等都可用来指定：应编写一条输入指令。
4. 在伪代码中，为了说明一系列对所有有效输入操作指令的重复出现，写成：

```
DO WHILE 还有输入数据
    :
    :
ENDDO
```

DO WHILE … ENDDO是伪代码表达式。"DO WHILE 还有输入数据"可能写成："DO WHILE 更多的数据"或"DO WHILE 非工作结束"，或其他表示同样意思的短语。

5. 伪代码中DO WHILE … ENDDO间的指令被反复执行，直至没有更多的输入要处理为止。这些指令的缩进是为了强调总体结构。
6. 条件测试在伪代码中可能写成

```
IF 条件  THEN
    :          ←  如果条件成立，执行这些语句
ELSE
    :          ←  如果条件不成立，执行这些语句
ENDIF
```

例1：计算平均分程序的伪代码

```
START
    输入三门考试成绩
    DO WHILE 还有更多的数据
        计算平均分数
        打印平均分数
        输入三门考试成绩
    ENDDO
```

```
STOP
```

例 2: 一个计算销售佣金程序的伪代码

```
START
    输入一个销售量
    DO WHILE 还有更多的数据
        IF 销售量>100.00 THEN
            销售佣金=0.10×销售量
        ELSE
            销售佣金=0.05×销售量
        ENDIF
        PRINT 销售佣金
        输入销售量
    ENDDO
    PRINT "End of Job"
STOP
```

§ 1.6　本章辅导

§ 1.6.1　本章小结

Turbo Basic 特性

1. Basic是最广泛使用的程序设计语言之一.
2. Basic是Beginner's All-Purpose Symbolic Instruction Code（初学者的通用符号指令代码）的缩略词.
3. Basic程序易读，易学，易编.
4. Turbo Basic是Basic的一个快速、高效、通用的微机版本，它利用窗口，使用户可同时查看程序及其输入、输出.

程序的准备和调试

1. 从系统分析员或用户手中获取程序指标（输入格式、输出格式，处理要求）.
2. 使用规划工具，如伪代码，在编程之前规划程序结构.
3. 编程.
4. 翻译或编译程序.
5. 用调试方法测试程序.
   a. 桌面检查程序.
   b. 修正语法错误
   c. 进行结构走查.
   d. 准备现实且完备的测试数据.
   e. 在程序执行之后检查逻辑错误.
6. 编写程序的文档.

设计良好程序的方法

1. 结构化程序设计
   a. 称作无GOTO程序设计.
   b. 结构化程序细分为程序段或模块.
2. 自顶向下程序设计
   a. 在小模块之前，编写主模块或主过程.
   b. 它类似于在编写报告之前先写大纲——组织和结构是最重要的；然后在小模块中填充详细内容.

§1.6.2 本章自测

在每章的末尾均有一个覆盖全章内容的自我测验，并附有解答.

1. 输入数据的另一个术语为________.
2. 输出信息的另一个术语为________.
3. 输入数据的两种主要方法为: 从________或________输入.
4. 简化Turbo Basic程序设计和方便调试的两种方法叫________和________.
5. （T或F）结构化程序有时称作无GOTO程序.
6. 一些Basic版本用________翻译，它将扫描代码和执行代码融为一体；而更有效的版本使用________翻译，先翻译后执行.
7. Turbo Basic使用的翻译程序叫________.

8. （T或F）Turbo Basic指令与大多数通用的解释型Basic版本兼容.
9. 描述所有类型程序的术语叫________.
10. ________监控计算机的总体操作.
11. （T或F）程序员以编写程序作为开始.
12. 两种最通用的程序规划工具为________和________.
13. ________程序由编译器翻译成________程序.
14. （T或F）编译器将检查逻辑错误.
15. Basic是________的缩略词.
16. （T或F）结构化程序比非结构化程序易读、易调试、易修改、易维护.
17. 结构化程序按程序块，即________编写.
18. （F或F）Turbo Basic要求有行号.
19. ________指令用来在屏幕上显示信息，而________指令在打印机上打印信息.
20. Edit窗口中显示什么?
21. RUN窗口中显示什么?
22. 程序开发过程中的步骤是什么?
23. 在Turbo Basic程序中用来反复执行一系列步骤的指令为________和________.
24. 指令"DO WHILE Amt<9999"中的9999称作________. DO WHILE … LOOP将反复执行，直至________为止.
25. （T或F）伪代码在程序编写完毕后准备.

§1.6.3 复习题

一、是非题

T（对）或F（错）

1. 编译时无任何错误的Turbo Basic程序总能正确运行.
2. 执行前，Turbo Basic程序必须转换为机器语言.
3. Basic是一种适用于微机和大型机的通用语言.
4. Turbo Basic程序在运行前必须编译.
5. Turbo Basic是一种编译型（而不是解释型）符号语言.
6. Turbo Basic是一种操作系统程序.
7. 程序中的逻辑结构必须在编码之前规划.
8. 在编码之前规划过的程序易于调试.
9. Turbo Basic中的注释用撇号或单引号作为开始.
10. 除非DO WHILE … LOOP或其他逻辑控制结构改变了流程，否则指令一般顺序执行.
11. Turbo Basic程序不要求行号.
12. 程序模块可单独编码和测试.
13. 正确的程序设计最好在主模块之前编写小模块.

14. 终止值用来标明无更多的数据要处理.

15. LPRINT指令用来在屏幕上显示数据.

二、综合题

1. 定义下列术语:
   (a) 程序
   (b) 编译器
   (c) 源程序
   (d) 目标程序
   (e) 翻译程序
2. 陈述符号程序设计语言和机器语言间的区别.
3. 陈述Turbo Basic 是一种通用语言的原因.
4. 结构化程序设计的意思是什么?
5. 语法错误的意思是什么?
6. 如何用伪代码描述DO WHILE … LOOP?
7. 在Turbo Basic屏幕上Edit窗口中显示的内容是什么?
8. 在Turbo Basic屏幕上RUN窗口中显示的内容是什么?
9. LPRINT指令与PRINT指令的区别?
10. 下面两条语句有何不同?

```
This program prints a sum of amounts
```

和

```
PRINT "This program prints a sum
```

§ 1.6.4 编程题

在后面的各章中，将要求读者调试 Turbo Basic 程序. 本章我们提供两个 Turbo Basic 程序，要求读者提供输入并得出相应的输出；也就是说，执行一个描述应出现在 Run 窗口中情况的结构走查. 下述程序是分别利用 22 页和 23 页的伪代码设计的.

1.
```
'This program reads in exam grades and calculates an average
PRINT "Enter three exam grades--type 999,999,999 when done"
INPUT Exam1, Exam2, Exam3
DO  WHILE Exam1 < 999
    LET Average = (Exam1 + Exam2 + Exam3) / 3
    PRINT "The Average is "; Average
    INPUT Exam1, Exam2, Exam3
LOOP
END
```

2.
```
'This program calculates sales commission
PRINT "Enter Sales Amount--type 99999 when done"
INPUT SalesAmount
DO  WHILE SalesAmount < 99999
    IF  SalesAmount > 100.00 THEN
        LET Commission = .10 * SalesAmount
    ELSE
        LET Commission = .05 * SalesAmount
    END IF
    PRINT "The Commission is "; Commission
    INPUT SalesAmount
LOOP
PRINT "End of Job"
END
```

第二章　Turbo　Basic 结构化程序设计

本章要求:

应熟悉如下内容:

1. 硬件和软件的概念.
2. Turbo　Basic 的命令结构.
3. Turbo　Basic 编辑.
4. Turbo　Basic 的程序编码.
5. Turbo　Basic 程序的运行.

本章首先向没有或很少用过计算机的读者提供一般的硬件和软件知识. 硬件是指一个计算机系统的部件或装置; 而软件是指能够使计算机运行, 执行所要求的任务的程序. 如果用户已经用过微型计算机, 可以直接阅读第三节.

注意, 建议读者以"手册"方式来学这章内容, 坐在一台正在工作、并装有 Turbo Basic 系统的计算机前, 按照每个步骤阅读本章内容, 同时检查提供的每一个例子; 这是熟悉 Turbo　Basic 环境的最好方法. 一旦知道怎样使用 Turbo　Basic, 以后各章就可以按正常方法阅读. 上机是检测读者理解程度的唯一手段.

§ 2.1　硬件的一般性知识

这部分我们考虑一台个人计算机一般的构成. 个人计算机也叫 PC 机或微型计算机. 我们着重讨论 IBM 系统列的个人计算机. 但是若有 IBM 兼容机, 读者将发现其设备与这里讨论的很相似. 我们将同时讨论 IBM PC 机和 PS／2 系列的 PC 机 (参见图 2.1 和图 2.2). IBM 微型机及其兼容机均能支持 Turbo　Basic . 假设计算机买来时就已经配备以下的装置:

1. 键盘 2. 显示器 3. 打印机 4. 至少有一个磁盘驱动器	输入 ／ 输出设备
5. 微处理器	中央处理单元

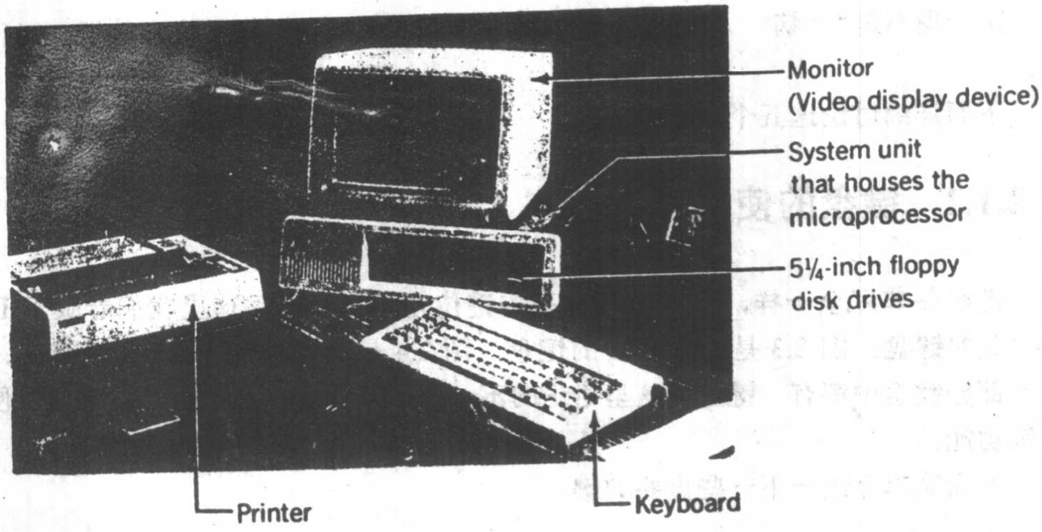


图 2.1　IBM PC

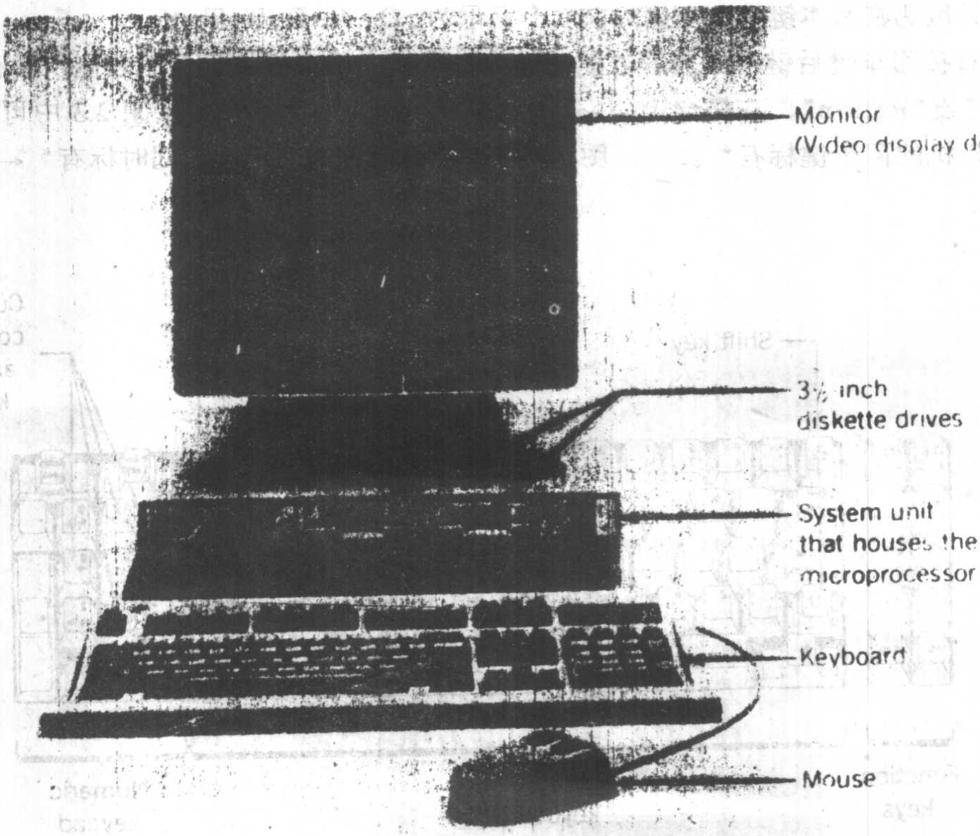


图 2.2　IBM Person System / 2 组件

微处理器是"大脑"，即一个系统的处理单元，它能够贮存程序和数据，运行所有的程序。

下面详细讨论这五个部分。

§2.1.1 键盘的使用

键盘并非完全一样，但大数键盘基本键位差不多一样。这里我们着眼于 IBM PC 和 PS／2 的键盘。图 2.3 是 IBM PC 的键盘，图 2.4 是 PS／2 的 IBM 高级键盘。下面讨论在大部分键盘中都有。键盘与这里讨论的不一样，读者应该熟悉自己的键盘以便学会一些特殊功能。

下面简单介绍一下一些重要的键。

§2.1.1.1 基本键盘

大多数微型计算机的基本键盘拥有与打字机类似的键，我们称之为 QWERTY 键盘，这是因为在基本键盘左上侧的前六个字母为：Q–W–E–R–T–Y。

只有按回车键后键入的内容才能传递给计算机，回车键在键盘的右侧，标有"↵"、"Enter"或"Return"。在概念上，它同打字机上的回车键相似。从图 2.3 中可以看到，IBM PC 机的回车键标有"↵"；图 2.4 中高级键盘上的回车键，同时标有"↵"和"Enter"。

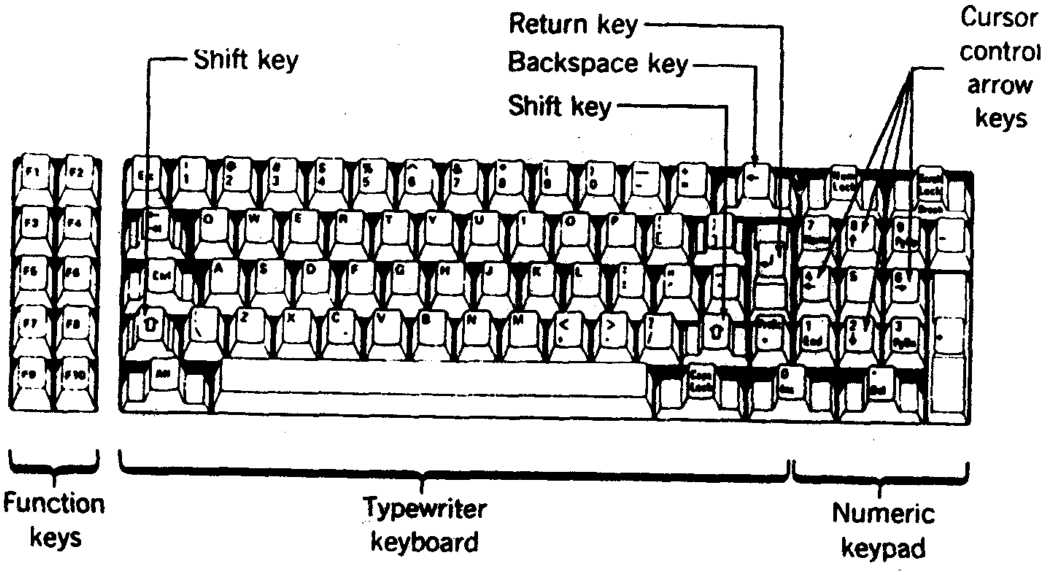


图 2.3 IBM PC 键盘

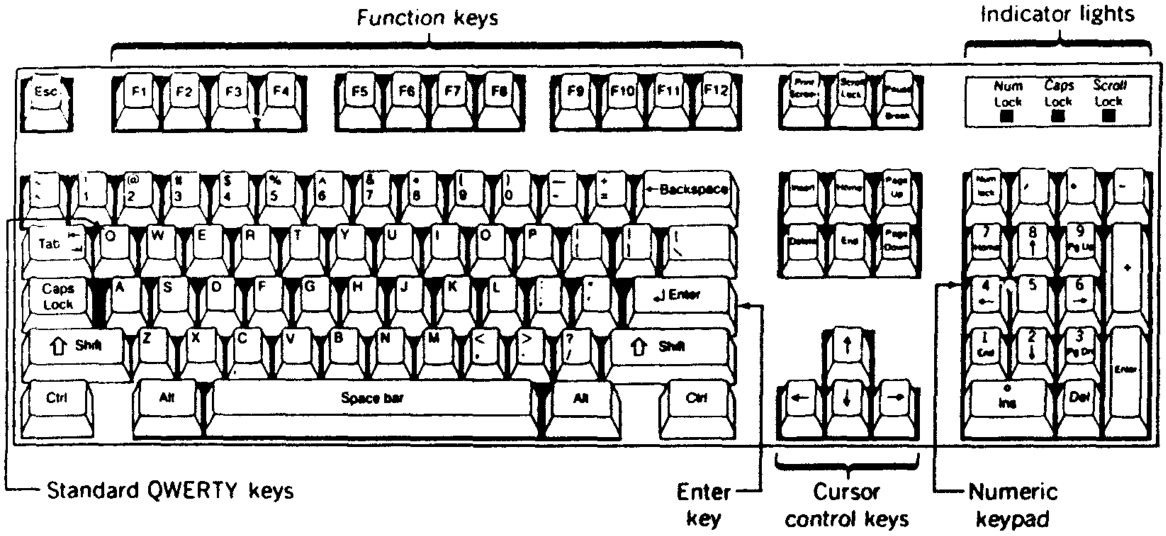


图 2.4　PS／2 上使用的 IBM 高级键盘

象打字机一样，可以用键盘输入指令和数据．准备在计算机换行时，键入 ↵ 或 Enter 键．在按下这个键之前，可以改变键入的任何东西，因为这时数据或命令没有传递给计算机．

如果在键入过程中发生错误，可以使用退格键（BackSpace）移回光标，擦掉字母．退格键标有“←”，在键盘的右边，位于回车键上方（参看图 2.3）．每按一次回退键，计算机将抹掉光标左边的一个字符．

除了回车键和退格键外，还应该熟悉其它的一些键．

§ 2.1.1.2　数字小键盘（Numeric keypad）

数字小键盘一般位于基本键盘的右边，参看图 2.5．数字小键盘可以使用户方便地用右后输入数字，而不必使用基本键盘上面的数字键．但在快速输入时，这些数字键显得不太方便．

光标控制键，如↓、↑、→、←、Home、End、PgUp、PgOn，一般标在数字小键盘上的数字下面．如果光标控制字符是在数字小键盘数字的下面，那么相应键时，它们首先有效．也就是说，如果按←键，光标就左移一列．若想用数字小键盘输入数字而不是被用作光标控制键，必须按一次 NumLock 键．从图 2.5 中可看到，该键位于数字小键盘的上面．高级键盘除了数字小键盘上的光标控制键外，还有一系列独立的光标控制键．

NumLock 键的功能象一个开关键．按下 NumLock 键后，为了能够使用光标控制键，而不是数字，应该再按一次 NumLock 键，可以用数字小键盘的计算机输入数字，按两次，又可以用数字小键盘控制光标移动．我们把具有这种功能的键叫做“乒乓开关键(toggle key)”．

如果 NumLock 键根本就没有按，那么按数字小键盘上的键时，就相应地控制光标的

移动.

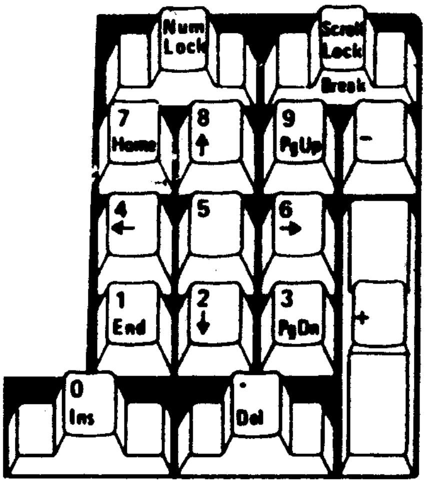


图 2.5 数字小键盘

§ 2.1.1.3 CapsLock 键

CapsLock 键位于 IBM PC 键盘的右侧，或位于高级键盘的左侧（参见图 2.3 和图 2.4）. 象 NumLock 键一样，也是一个乒乓开关键. 如果根本不按它，那么按字母键时，输入计算机的是小写字母；若按一次该键，便可以向计算机输入大写字母；再按一次，又变成小写字母状态. 象 NumLock 、CapsLock 这样的乒乓开关键，有时有指示灯清楚地显示它们是"开"（按一次），还是"关"（按偶数次）.

§ 2.1.1.4 Shift 键

CapsLock 键只对字母起作用，而不影响键盘顶行的数字和特殊符号. 例如，为了输入基本键盘数字 8 上面的 * 号，必须使用标有"↑"的换档键（Shift key）. 键盘上的两个 Shift 键分别位于基本键的左下角和右下角. 使用 Shift 键可输入键盘上的特殊字符，而且还可以输入大写字母. 但是 Shift 键不是乒乓开关键，因为每次输入大写字母和特殊字符时，都必须按 Shift 键. 而 CapsLock 键则只需按一次，即可输入所有的大写字母. Shift 键与 CapsLock 键的功能与它们在打字机上的功能相同.

§ 2.1.1.5 光标键

由图 2.5 可注意到，光标箭头（键）一般出现在数字小键盘作为控制字符. 它们的功能是把光标定在屏幕上所要求的位置. 通过使用↑、↓、←、→光标控制键，可以使光标位于屏幕上的任何位置. 在 Turbo Basic 中，光标控制键可以使要执行的命令高亮度显

示，也可以在输入程序时使用；用户可以使用这些键在屏幕上的任何地方输入程序．在大多数程序和软件包中，用这种方法指明特殊命令或修正数据．

对于 Turbo Basic 和许多其它软件，位于数字小键盘上的 PgDn 键可以使程序或文件往下翻页，而 PgUp 键可以往上翻页．同样，位于数字小键盘上的 Home 键可以使光标置于一行的最前面，而 End 键使光标移到一行的最后面．

注意，高级键盘除了在数字小键盘上有一套光标控制键外，还有一套单独的光标控制键．

§2.1.1.6 功能键

对于 IBM PC 键盘，功能键 F1～F12 出现在主键盘的左侧（参看图 2.3），在图 2.4 中，IBM PS／2 高级键盘的功能键 F1～F10 位于主键盘的最上边．功能键对于每一个程序包有其特殊意义，例如在 Turbo Basic 中，F1 功能键用于显示一幅帮助屏幕（Help Screen）．每一个程序包使用功能键均有其目的．功能键通常的编号是 F1～F10 或 F1～F12．以后，我们将给出它们在 Turbo Basic 中的用法．

§2.1.2 监视器（Monitor）

用键盘输入数据或计算机作出响应时，被键入的数据和计算机的结果都将在监视器上显示出来．监视器又叫屏幕，CRT 或视屏．一些监视器有自己的“ON”开关．若监视器上有一上开关或旋钮标有“ON／OFF”或“I／O，那么在使用之前首先应把它打开．

有些监视器是单色监视器，它们只能显示黑白或琥珀色或白色．而其它的是彩色监视器．为了能在屏幕上显示图形和色彩，计算机必须有一个图形适配器．

监视器通常带有一个调节屏幕颜色或亮度的旋钮，用户可以按照自己的爱好调节．

对于一台微型机，可以使用许多种彩色图形屏幕(在第十四章考虑图形结构时详细讨论)．

大多数监视器可显示二十五行，每行八十列．

§2.1.3 打印机

打印机用来获得程序或屏幕上数据的一个硬拷贝．所有打印机都能用黑油墨打印字母，数字和特殊字符，而许多打印机也能够进行彩色打印和打印图形字符．

打印机有点阵打印机和优质字符打印机两种，许多点阵打印机在清晰度上与优质字符打印机接近．

大多数打印机每行可以打印 80 个字符，但也有一些打印机每行能打印 100 或 120 个字符．打印机有单独的“ON”开关．

回送打印（Echo Printing）：为了能够在 DOS 或 Turbo Basic 状态下获得任何屏幕的输出的打印拷贝，同时按 Shift（上档键）键和 Prtsc 键（Prtsc 键位于键盘右侧）．

同时按下这两个键时，打印机将打印当前屏幕上所显示的内容，在 Turbo Basic 下使用回送打印时，必须使用图形打印机；这种打印机能够打印 Turbo Basic 屏幕中的框，否则，所打印框的轮廓将是一些相当奇怪的字母或符号.

为了从某一特定点开始回送打印，同时按下 Ctrl 键和 Prtse 键，这样就从那一点开始打印，与同时按 shift 键和 Prtse 键相对照，后者将打印屏幕上的所有东西.

§ 2.1.4 磁盘驱动器

§ 2.1.4.1 使用磁盘的原因

为了运行一个程序，这个程序文件必须首先输入到中央处理单元（CPU）中. 同样，那些生成的，读入的和现有的数据文件也必须先由 CPU 读入才能进行处理. 但是这些数据文件必须存在一些辅助的或者次存贮设备中. 这样，关机后，它们仍能够在将来的处理中重新使用. 我们经常将数据和程序按文件（file）形式存储在磁盘上; 它们可以再次使用.

§ 2.1.4.2 微型机上使用的磁盘种类

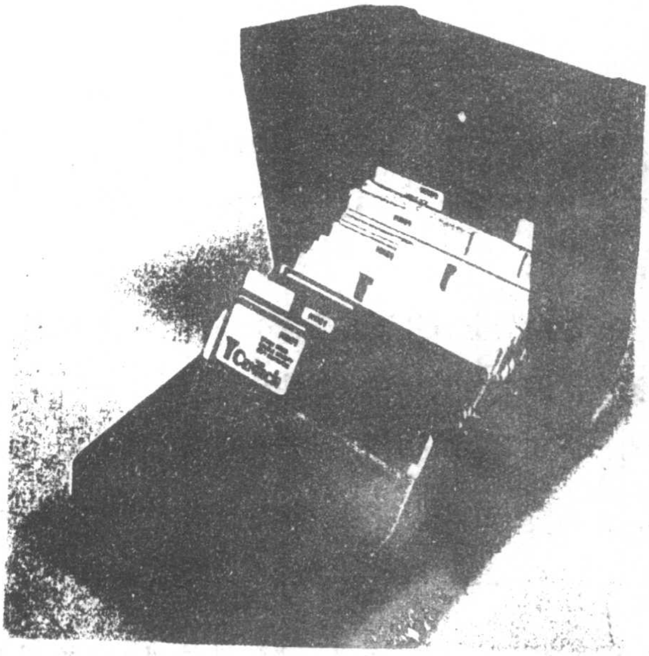
微型机使用两种磁盘:

(1) 软盘;
(2) 硬盘

虽然有只有一个软盘驱动器的计算机，但典型的软盘系统是将两个软盘驱动器用作辅助存贮器，而典型的硬盘系统有一个硬盘和一个或两个软盘驱动器.

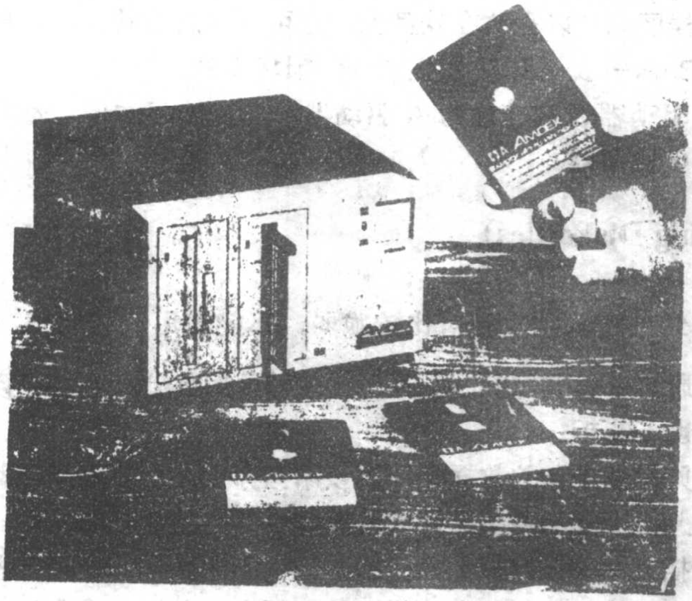
软盘驱动器可以作为微机的一部分或者是完全独立于微机的单元. 大多数 IBM PC 机使用$5\frac{1}{4}$英寸软盘；而 IBM PS／2 机，则使用$3\frac{1}{2}$英寸盘(参见图 2.6). 注意，$3\frac{1}{2}$英寸盘的存贮量要比$5\frac{1}{4}$英寸盘大.

$5\frac{1}{4}$英寸盘的存贮量大约在 80,000 个字节（80K）到 1.2 兆个字节（1.2MB）之间（实际上，1K＝1024 个存贮单元，所以 80K 比 80, 000 个存贮单元稍多一点）. 大多数$5\frac{1}{4}$英寸盘能够存贮 360K 个字节，而新一代软盘可以达到 1.2MB 或更多. $3\frac{1}{2}$英寸盘的容量在 720KB 到 1.4MB 或更高. 除了高容量外，$3\frac{1}{2}$英寸盘的其它一些优点包括:



(a)

图 2.6(a) 大多数 IBM PC 机上使用的$5\frac{1}{4}$英寸软盘



(b)

图 2.6(b) 在 IBM PS / 2 机和大多数便携机上使用的$3\frac{1}{2}$英寸软盘

$3\frac{1}{2}$英寸盘的优点

1. 由于其体积小，所以所需要的存放空间也少.
2. $5\frac{1}{4}$英寸盘有一个薄保护套，而$3\frac{1}{2}$英寸盘有一个硬塑料盒，这样更不易有刻痕.
3. $3\frac{1}{2}$英寸盘有一滑动的金属片覆盖磁性介质. 当磁盘放入驱动器，金属片自动滑出，以减少毁坏的可能性.
4. $3\frac{1}{2}$英寸盘有一固定的写保护. $5\frac{1}{4}$英寸盘也有可以粘贴的写保护，但很容易掉，而$3\frac{1}{2}$英寸的盘只要滑出调节片即可进行写保护.

与$3\frac{1}{2}$英寸和$5\frac{1}{4}$英寸软盘相比，硬盘(hard dish) 是一种具有多种用途的存贮装置，其典型容量是 10MB 到 80MB 之间. 图 2.1 是一台有一个硬盘驱动器的计算机. 除了大容量外，硬盘的另一个优点是它被一个盒子罩着，这样可以减少灰尘和处理不当等的影响. 硬盘可以用来存贮程序和数据，这样可以减少软盘交替插入驱动器的需要.

有一些微型计算机用的是硬卡而不是硬盘，象微处理器一样，硬卡只能暂时存贮数据，其目的和功能与硬盘一样.

注意，数据和程序既可以存放在软盘上，也可以存放在硬盘上. 任何数据和程序对文件都应有一备份 (Back-up)，以防发生意外，损坏文件.

左边或上边的驱动器通常被称为 A 驱动器，右边或下边的驱动器被称为 B 驱动器，硬盘驱动器被称为 C 驱动器.

软盘保管(Handling Diskettles)

软盘，特别是$5\frac{1}{4}$英寸软盘，相对比较脆弱，不能滥用；它们有一个外壳保护，绝不允许把外壳拿掉；通过外壳拿磁盘，千万不要接触金属部分. 软盘应该远离磁场，如电视机，电话和微波；也不能折.

用空白软盘拷贝程序或存贮数据时，应该贴上标签；用油笔在标鉴上注明内容. 磁盘插入时应该标签朝上，千万不要折磁盘或触摸金属部分.

为了把一块盘插入 A 或 B 驱动器，必须先打开驱动门，对于一些驱动器必须按释放键才能打开. 对于大多数驱动器而言，使用前必须先关上门. 磁盘驱动器对应的指示灯亮着时，表示它正在工作，千万不要在此时将磁盘插入驱动器，否则将损坏磁头.

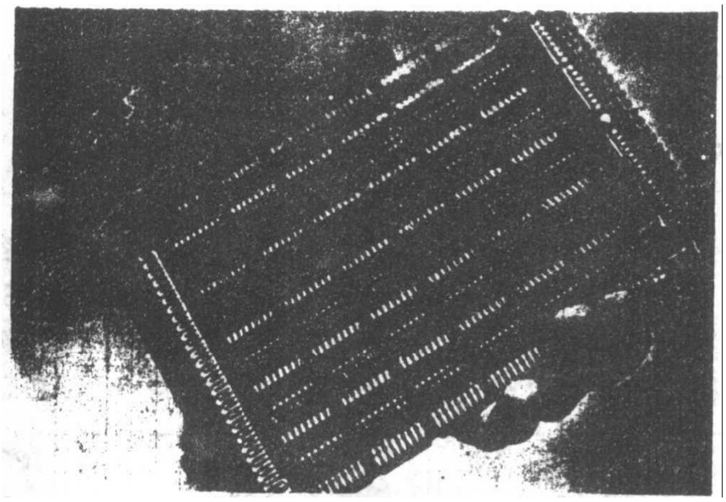
如前所述，软盘可以按如下方法进行物理性保护以防止意外擦写或重写:

$5\frac{1}{4}$英寸盘：把磁盘随带的金属标鉴贴在写保护槽以防止重写。

$3\frac{1}{2}$英寸盘：作为磁盘一部分的金属片置于写保护槽上面可以防止写入。

§2.1.5 微处理器

微处理器是计算机系统的大脑，被称为中央处理单元（CPU- Central Processing Unit）安装在主机的一块或几块板上。它由芯片构成，含有上千个数字电路——参见图 2.7(a)和图 2.7(b)。



(a)

图 2.7(a) 一块面积小于 1／4 英寸，集成上千个电路的集成电路芯片

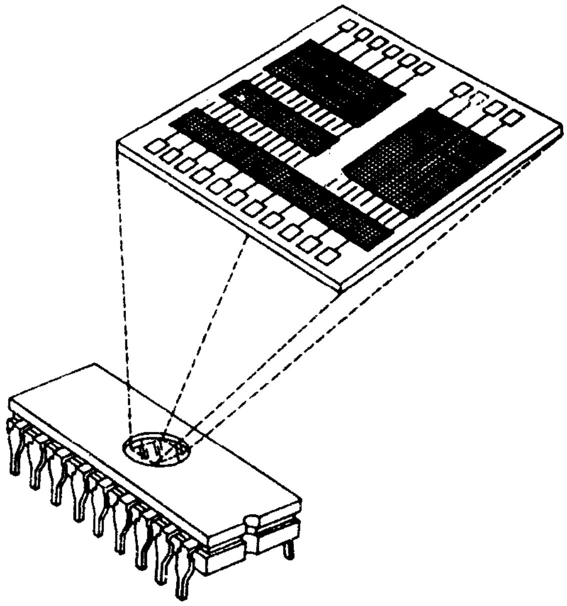
微处理器存贮所有数据，处理所有指令。它由两个存贮器构成：

(1) 只读存贮器（ROM），它是为执行系统功能而事先配置存贮器。ROM 由生产厂家安装，用户自己不能更换。

(2) 随机存取存贮器（RAM）用于存贮程序和数据。

对于大多数微处理器，RAM 常用的容量单位是千字节（kilobytes）；1K 字节等于 1024 字节。虽然可以用存贮器扩充卡，但常用的 RAM 容量范围是 256K 到 640K 字节，使用存贮器扩充卡后，一些微处理器的存贮容量可超过 1 兆字节（megabyte）。

选择微处理器时，主存贮器容量（primary storage capacity）是需考虑的最重要因素之一。一台微处理器的最大存贮能力必须预先知道，即使最初得到的是其内存小于最大内存的"初级"系统。注意，Turbo Basic 需要至少 320K 的存贮能力。



(b)

图 2.7(b) 各集成片均装入带插脚的保护壳内

微处理器中使用的芯片类型一般是 8086 系列. 标准的 PC XT 及其兼容机用的都是 8086 芯片; PS/2 的 25 型和 30 型也是用 8086 芯片. IBM AT 和 IBM PS/2 50 和 60 型计算机使用一种高级的芯片 80286. 在 IBM PS/2 70、80 型和其它高级微型计算机中, 使用的是更高级的芯片 80386. 下表提供了一些不同微处理器芯片的信息.

	PC XT, PS/2 25 和 30 型	AT、PS/2 50 和 60 型	PS/2 70 和 80 型
微处理器	8086	80826	80386
标准存贮容量	640K	1MB	2MB
扩充容量	640K	7~15MB	16MB
扩充端口	3 个	3~7 个	7 个

一些计算机有一个 8087 协理器, 用来提高数学运算速度. Turbo Basic 完全支持这种协理器.

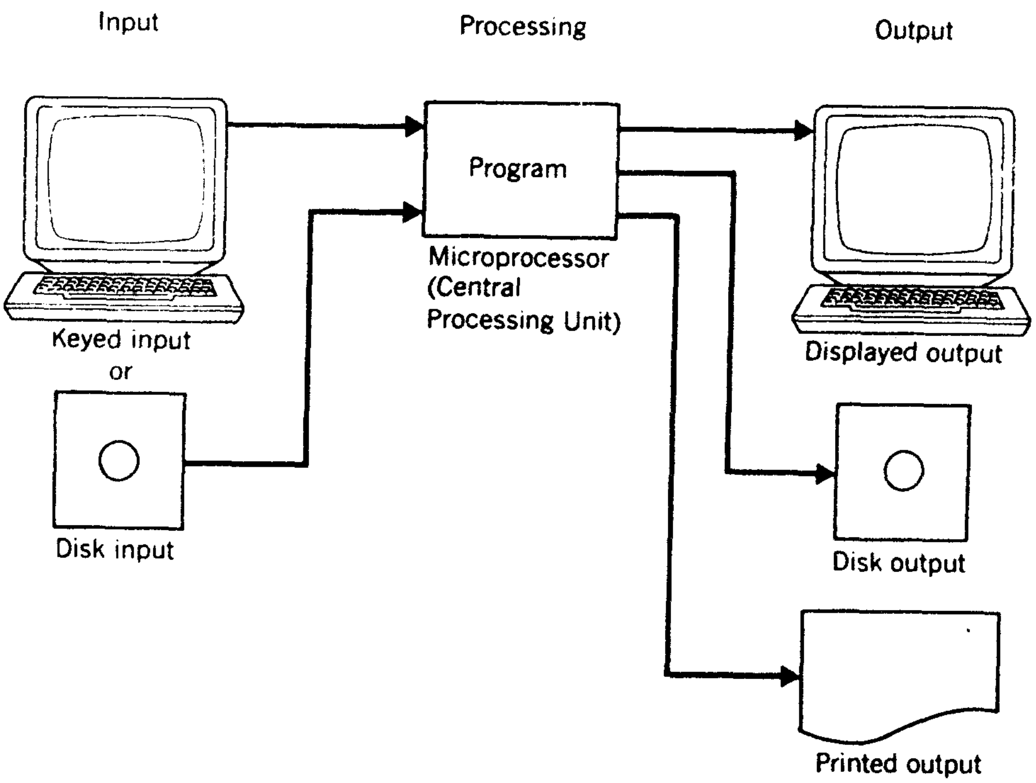
§2.2 软件介绍

软件是计算机程序的总称. 程序是一系列能够使计算机完成给定任务的指令. 软件主要有两种: 操作系统软件和应用软件. 操作系统软件用于管理计算机的整个运行; 它通常随机带来, 也可以单独向软件商购买.

应用软件可以是为许多用户设计的通用程序包, 也可以是为单用户编写的程序. 用

Turbo Basic 编写程序时，所编写的将是用户应用程序.

计算机处理数据的过程如下：



操作系统控制整个计算机的操作：

注意，一个操作系统是所有能够监视一台计算机整个操作，以及帮助用户有效使用计算机资源的程序总称. 启动计算机时，必须在磁盘驱动器里有操作系统，否则计算机将不能运行.

IBM PC （及其兼容机）和 PS／2 微机通常使用下面两种操作系统中的一种.

常用的操作系统

名字	全称	特征
DOS	Disk Operating system	MS－DOS 和 PC－DOS 是磁盘操作系统的两个版本，它们是 IBM 及其兼容机上最常用的操作系统
OS／2	Operating system 2	一种专门为 PS／2 系列微机设计的复杂操作系统.

用户必须学会一些操作系统的基本命令以便启动计算机．注意，这里只是对这些指令作个介绍，详细的讨论超出本章范围．这里涉及到的指令对 DOS 和 OS／2 都是有效的．本书将提供足够的细节以便帮助读者启动操作系统．这样，就可在操作 Turbo Basic 时使用．

§ 2.3 访问计算机的方法

§ 2.3.1 开机

每台微机都有一些硬件，可能有一个键盘和监视器，还可能有一台打印机或其它的一些设备．如鼠标，光笔或调制解调器．一般有一个或两个软盘驱器．同时，可能还有一个硬盘驱动器．如果有两个软盘驱动器，左边或上边的那个是主要的，通常称之为 A 驱动器，另一个称为 B 驱动器；如果有一个硬盘，就称之为 C 驱动器．

为了启动微机，必须先打开微机电源，“ON”开前在微机的后边，侧边或前面．

在开机之前，应确定是否有一个硬盘驱动器或者是硬卡．

对于只有软盘驱动器的微机，应该先把操作系统盘插入 A 驱动器中，然后开机；对于有硬盘的计算机，只要 A 驱动器中没有软件，微机将直接通过 C 驱动器启动．

§ 2.3.2 装入操作系统

以下几条有助于计算机的启动:

起　　动

1. 准备好操作系统
   a. 若只有软盘驱动器，把有操作系统的软盘插入A驱动器中．
   b. 若使用硬盘起动，则在启动计算机之前，硬盘就应该有操作系统．
2. 打开计算机
3. A驱动器的红灯亮，表示磁盘驱动器正在读盘，注意：不要在驱动器指示灯亮时拿开或更换驱动器中的磁盘．若 A 驱动器中有软盘，计算机则先处理它，否则，计算机就读硬盘．因此，装有操作系统的软盘必须插入 A 驱动器中或把操作系统装入硬盘中．
4. 大多数操作系统自动执行硬件诊断，以确保硬件正常工作．
5. 若硬件有问题，将得到警告；若一切正常，计算机将给出一些信息，并且给出一个提示 (propt)．这个提示需要用户回答．例如，使用 DOS 系统将出现下面的信息与提示．

```
Current data is Tue 1-01-1980
Enter new dtat:  [提示]
```

6. 得到如下提示时:

A>　(软盘操作系统)

或

C>　(硬盘操作系统)

这就表示，计算机准备工作.

用这种方法启动计算机系统被称为"冷起动".

时间和日期

1. 日期

使用DOS操作系统时，计算机将提示用户"Enter new data"，日期输入的形式为月-日-年，并且均为两位数，所以 1990 年 2 月 27 日应为 02-27-90.

2. 时间

时间输入的格式为: hh: mm: ss: mm

hh—小时
mm—分
ss—秒
xx—秒的小数位

注意: 小时和分钟、分钟和秒之间用冒号(:)分隔;
　　秒和百分秒之间用小数点(.)分隔;

DOS时间是24小时制，所以早上1时到中午是01到12，下午1时为13时，下午2 时为 14 时，等等. 例如，若想输入下午 1: 06，格式则为:

Enter new time : 13:06:00.00

或

Enter new time : 13:06

例

```
Current date is Tue  1-01-1980
Enter new date (mm-dd-yy): 11-03-88  ←—— User-supplied date
Current time is  0:00:47.28
Enter new time: 12:00  ←—— User-supplied time

The COMPAQ Personal Computer MS-DOS
Version 3.00

(C) Copyright COMPAQ Computer Corp. 1982, 1983, 1984, 1985
(C) Copyright Microsoft Corp. 1981, 1982, 1983, 1984, 1985
```

C>　←　C 提示符

虽然可以按回车键的办法省略回答时间和日期，然而，输入时间和日期是必需的．因为计算机将记录建立一修改程序的时间和日期．这样，目录中就有程序的日期和时间，以便区分备份文件或源文件．

记住，在输入日期和时间之后应按回车．

若计算机内部有时钟，可以设置系统使其省略日期与时间的提示．

练习题

1. 找到微机开关
2. 把系统盘插入A驱动器中。
3. 引导系统
4. 回答所有提示．

§2.3.3 计算机系统的重新引导

任何时候，用户若要重新起动计算机，我们称之为“重新引导 (rebooting)”．此时，只需同时按 Ctrl，Alt 和 Del 三个键，系统便自动重新启动，显示上节所述的日期提示．

注意，也可以用关机后等待 45 秒钟左右再开机办法重新启动计算机．这时要求，对于软盘系统，保证在 A 驱动器有操作系统盘，对于硬盘系统，A 驱动中可以没有磁盘(硬盘中应含有操作系统)．通过先关机然后开机的方法重新启动计算机，叫做“冷启动”；与此相应的的，称为“热启动”．冷启动需要较长时间，因为要等一分钟后才能再次开机．

若读者已知道怎样进行一个软件包的工作拷贝，可以略过下节．

§2.4 准备一份 Turbo Basic 软件的工作拷贝

由 Borland International 公司发行的 Turbo Basic 装在一块$5\frac{1}{4}$英寸的软盘中，这块盘叫 Turbo Basic 程序盘，另外一块$5\frac{1}{4}$英寸装有 Turbo Basic 的示范程序．Turbo Basic 也可以装在一块$3\frac{1}{2}$英寸盘中（$3\frac{1}{2}$英寸盘的存贮容量一般比$5\frac{1}{4}$英寸盘的大，一块$3\frac{1}{2}$英寸盘能装两块 $5\frac{1}{4}$英寸盘的内容）．

Turbo Basic 源程序盘的内容应该拷贝到一块工作软盘或硬盘中．一旦拷贝完毕，应把源盘放在字全的地方保存．一般情况下只用它来拷贝工作盘，为了防止因疏忽而偶然对源盘进行写操作，应对源盘写保护．

有些程序软盘是拷贝保护 (copy protected) 的，而 Turbo Basic 不是．下面分两部

分讨论如何制作工作拷贝:

(1) 用不带硬盘的两个软盘驱动器（软盘系统）;
(2) 用一个硬盘和一个软盘驱动器把软盘拷到硬盘上.

§2.4.1 软盘系统（A 和 B 驱动器，没有硬盘）

准备工作拷贝的一个方法是:

1. 把操作系统盘（DOS 或OS／2）插入A驱动器中
2. 恢复没有使用过的新盘. 把写有软件名（如Turbo Basic−工作拷贝）的标签贴到磁盘上，然后把空白磁盘放入 B 驱动器中，在使用之前，必须首先对它进行格式化.
3. 打开计算机，回答日期和时间.
4. 出现 A提示符后，输入: FORMAT B:（不要忘记冒号）.

此时，计算机将出现如下提示:

```
Insert new dikette for drive B:
and strike Enter when ready
```

确信 B 驱动器中有空的磁盘后，按回车键，随即 B 驱动器提示灯亮并显示:

```
Formatting …
```

一会儿后，计算机将显示格式化完毕，并一提供一些详细信息，如软盘的存贮空间，并询问是否还要对另一软盘格式化，

```
Format another (Y／N) ?
```

这时，不需要拷贝 Turbo Basic 示范程序盘，所以打 N 表示 NO（不拷贝）.

我们不能把操作系统拷到这块目标盘中，因为对大多数软盘没有足够的空间同时装下操作系统和 Turbo Basic.

现在，可以把 Turbo Basic 程序盘，包括编译，编辑的另外一些文件拷贝到格式化后的空白目标盘上. 把 Turbo Basic 程序盘放在 A 驱动器中，目标盘放在 B 驱动器中，输入

```
COPY A:*. * B:
```

在 Turbo Basic 程序盘中有许多程序和信息文件，在拷贝过程中，文件名将被显示. 当所有文件拷贝完后，计算机将作如下的反映. 假设 Turbo Basic 程序软盘有 6 个文件:

6 file (s) copied

(根据不同的版本，Turbo Basic 软件有不同数目的文件.)

COPY 命令中的 *.* 表示要拷贝所有文件. 拷贝文件与源文件相同的文件名，此外，还可以按下面方法制作工作拷贝.

1. 把操作系统盘插入A驱动器中，开机
2. 键入命令: Diskcopy A: B:
3. 把系统务从A中拿出.
4. 把Turbo Basic 源程序盘放入A驱动器中，另外一块没有格式化的软盘放入 B 驱动器中.
5. 按回车键.

Disk copy 命令将对 B 盘进行格式化，同时把所有 Turbo Basic 源程序拷到 B 盘中.

§2.4.2 硬盘系统

启动后为了工作拷贝，把 Turbo Basic 盘放入 A 驱动器，键入:

```
C> md\tbasic
C> cd\tbasic
C> copy A:*.* C:
```

这样就可以把 Turbo Basic 程序盘中的程序拷到 tbasic 子目录下. 在硬盘中单独开一个子目录的目的是为了把相应的程序和数据放在一起作为一个单元. 由于硬盘能存贮上千万个字符，所以为了方便，开一个子目录是必须的. 也可以不把子目录称为 tbasic，若已经如此，那么所有 Turbo Basic 程序放在一个叫 tbasic 程序的区域中.

§2.5 Turbo Basic 程序概述

Turbo Basic 程序工作拷贝中一般包含如下一些文件:

README	COM
TB	EXE
TBINST	COM
TBHELP	TBH
HANOI	BAS
README	

为了能够使用包含编辑器和编译器的 Turbo Basic 系统，至少要有以下三个文件:

```
TB              EXE
TBINST          COM
TBHELP          TBH
```

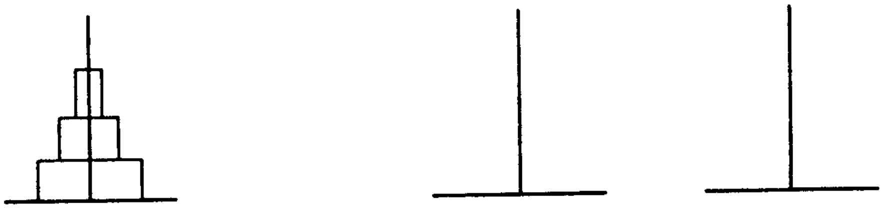
许多软盘有 README 文件，其内容是有关软件包或程序的信息，包括最近的更动，删除和添加，在使用之前最好看一下 README。文件名后部的.COM 或 EXE 表示该文件为可执行文件。README.COM 文件将 README（没有扩展名）数据文件作为输入。键入：

```
README
```

时，可以重新看一下 README 文件内容。

为了得到 README 文件的硬拷贝输出，在敲 README 之前先按 Ctrl 和 Prsc 键。

另一个程序是 HANOI.BAS，这是 Turbo Basic 的一个示例程序，用户可以编译运行这个程序。它是 Hanoi 塔的一个示例；把不同大小的盘从一个柱搬到另外两个柱中的任意一个，并不允许把大盘放在小盘上面。



目的：把盘从一个柱挪到另一个柱，整个过程不允许有一个大盘放在小盘上面

根据不同版本，除该示例程序外，还有另外一些文件示例程序。读者可以象运行自己的程序一样运行它们．下面几节说明怎样装入和运行 Turbo Basic 程序。

§2.5.1 进入 Turbo Basic

为了进入 Turbo Basic 系统（Turbo Basic system），根据所用的是软盘系统，还是硬盘系统，进行如下操作：

1. 软盘系统（没有硬盘）
   a. 将系统盘插入A驱动器中。
   b. 引导系统，回答日期和时间提示。
   c. 把Turbo Basic 工作拷贝放入A驱动器中，在A提示符下打TB：

```
A>TB
```

2. 硬盘系统:

a. A驱动器中不插入软盘，引导系统，回答日期和时间提示

b. 如果Turbo Basic系统在硬盘中，并且没有开子目录，在C 提示符下键入TB:

C> TB

若有一个 Turbo Basic 子目录，键入:

C>cd＼ (子目录)
C>tb

例:

C> cd＼tbasic
C> tb

不管是使用磁盘系统还是软盘系统，用户均可以键入小写 tb、大写 TB、甚至混合写 Tb 进入 Turbo Basic 系统；至此，可以使用 Turbo Basic 编辑器。

自我测验(一)

1. “Micro”是________的缩写
2. “micro”的另一个术语是________.
3. 微机的主要软盘驱动器叫做________驱动器.
4. 在开机之前，在主驱动器中应有________. 除非有一个硬盘，若有一个硬盘，强以通过________方式启动.
5. (T或F) 键盘中有两套键可以输入数字.
6. 在数据传送给计算机之前,必须按________键.
7. IBM 微机上最常用的操作系统是________.
8. DOS 系统一开始问用户________和________.
9. 要把一块软盘的内容拷到另一软盘时，使用________命令.
10. (T或F) diskcopy 命令自动格式化软盘而copy 命令无此功能.

§ 2.5.2 Turbo Basic 屏幕

为了进入 Turbo Basic，必须先装入操作系统，回答提示，然后再键入 TB，并按回车才能将 Turbo Basic 系统调入内存.

§ 2.5.2.1 初始屏幕

图 2.8 是 Turbo Basic 起始屏幕。

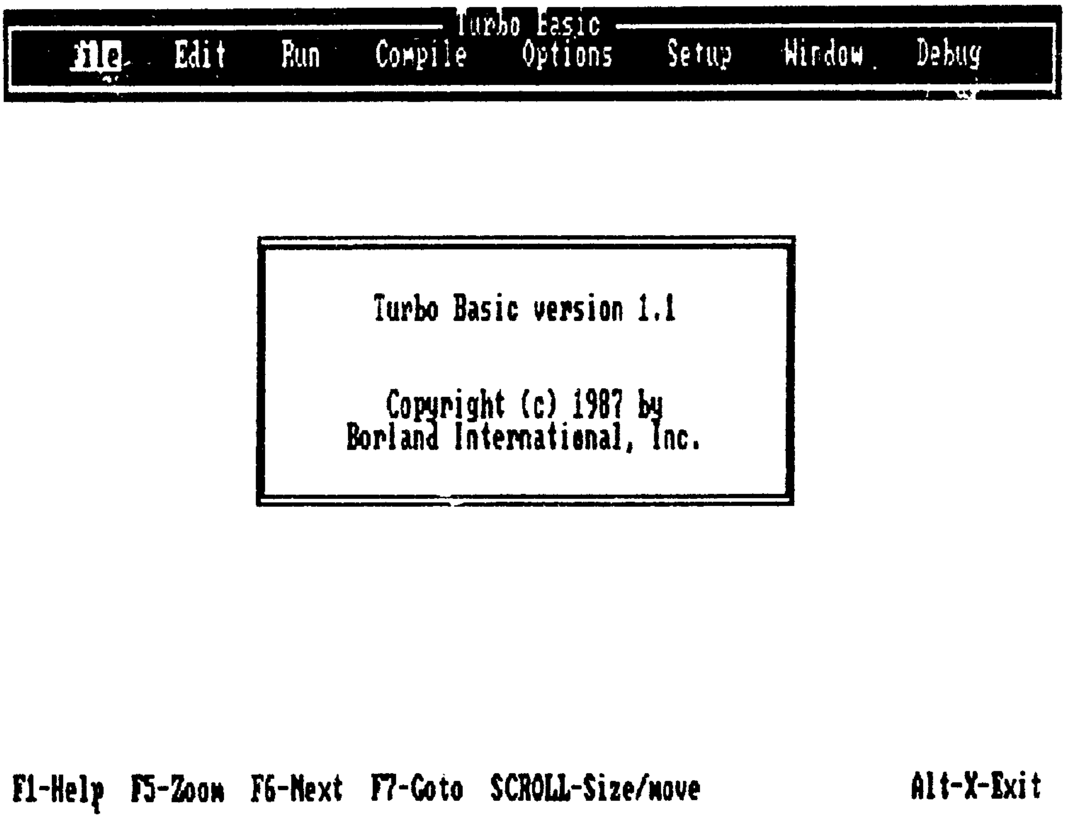


图 2.8 Turbo Basic 初始屏幕

在阴影区域里的字是命令。当屏幕一开始出现时，File 命令的字母反相显示非常显眼，(例如：在图中，File 命令黑字白底，其它命令是白字黑底)．Turbo Basic 的版本可能是 1.0、1.1 或更高，2.0 版是最新的，屏幕最底行是帮助行（Helpline）．

§ 2.5.2.2 主屏幕

此时若按回车键，Turbo Basic 整套窗口出现在主屏幕上（如图 2.9 所示）：

在阴影命令下面有四个窗口分别称作 Edit 、Trace 、Message 和 Run 。此时，又有一些键及其功能列在主屏底行，这一行称为帮助行（Helpline)，(以后详细讨论)．

下面讨论编辑和调试 Turbo Basic 程序所必需的命令和选项．以后几章介绍另外的一些命令和它们的选项。第九章着重于怎样设置屏幕以满足用户要求，此外还提供许多有关的 Turbo Basic 系统的高级特性。这样可以使用户最大限度地利用这一软件的优点。

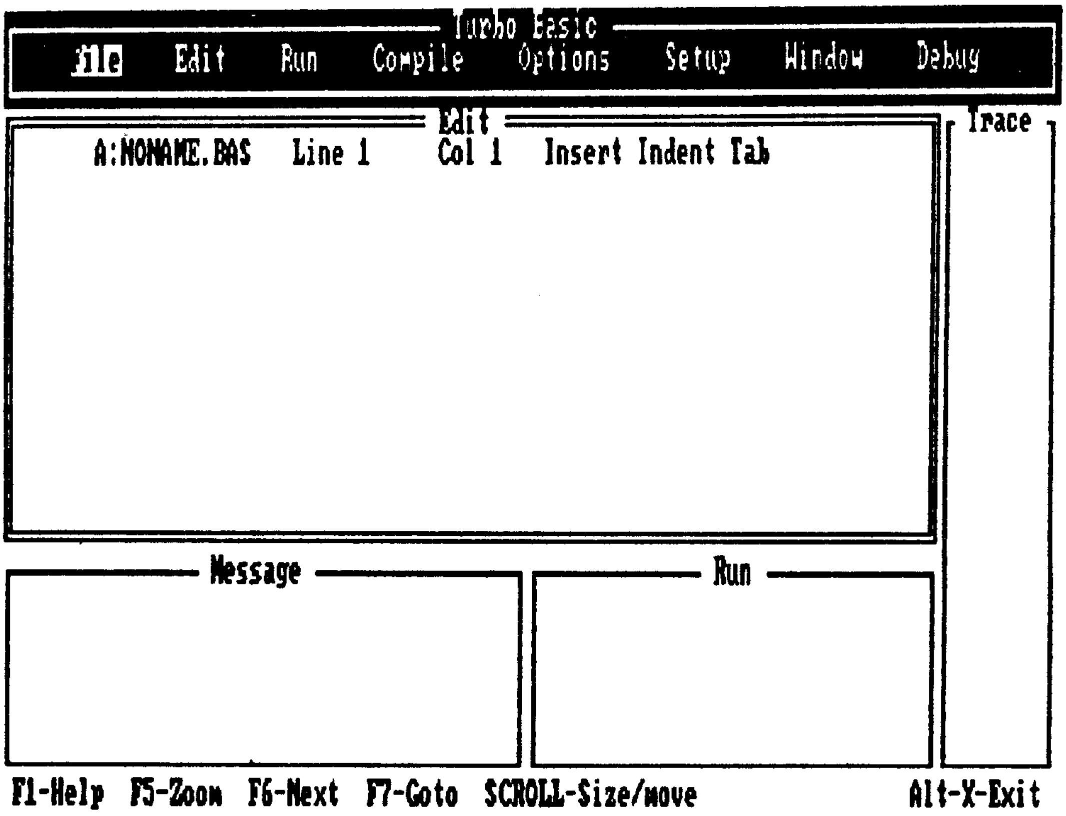


图 2.9 Turbo Basic 主屏幕

§ 2.5.2.3 主屏窗口

图 2.10 描述每个窗口的功能．

a. Edit窗口：显示Turbo Basic 程序．

b. Run窗口：显示输入、输出

c. Message窗口：当编译和运行一个程序时，该窗口提供一些有关编译和运行的信息．它将显示程序的行数，被编译程序的大小，所剩的内存等等．当所编写的程序大小接近于有效的内存体积时，这样的信息是非常有用的．也就是说，它们可以警告用户：一个程序太大，以致不能装入现有的内存．

d. Trace 窗口；如果每个指令均有行号，Trace 窗口可以列出程序执行时的行号．这样，由于一个逻辑错误引起程序中断时，用户可知道哪一条是最后执行的命令．第八章将对如何最有效使用 Trace 选项提供更深入的介绍．图 2.11 是第一章讨论过的示例程序．它也描述了选取 Trace 选项时，Trace 窗口将如何列出在执行程序时遇到的行号．在图 2.11 中列出了行号顺序假定有两个输入数据行和一些终止行（其值为 STOP，999，999），如 Run 窗口所示．下面是对 Trace 窗口条目的解释．

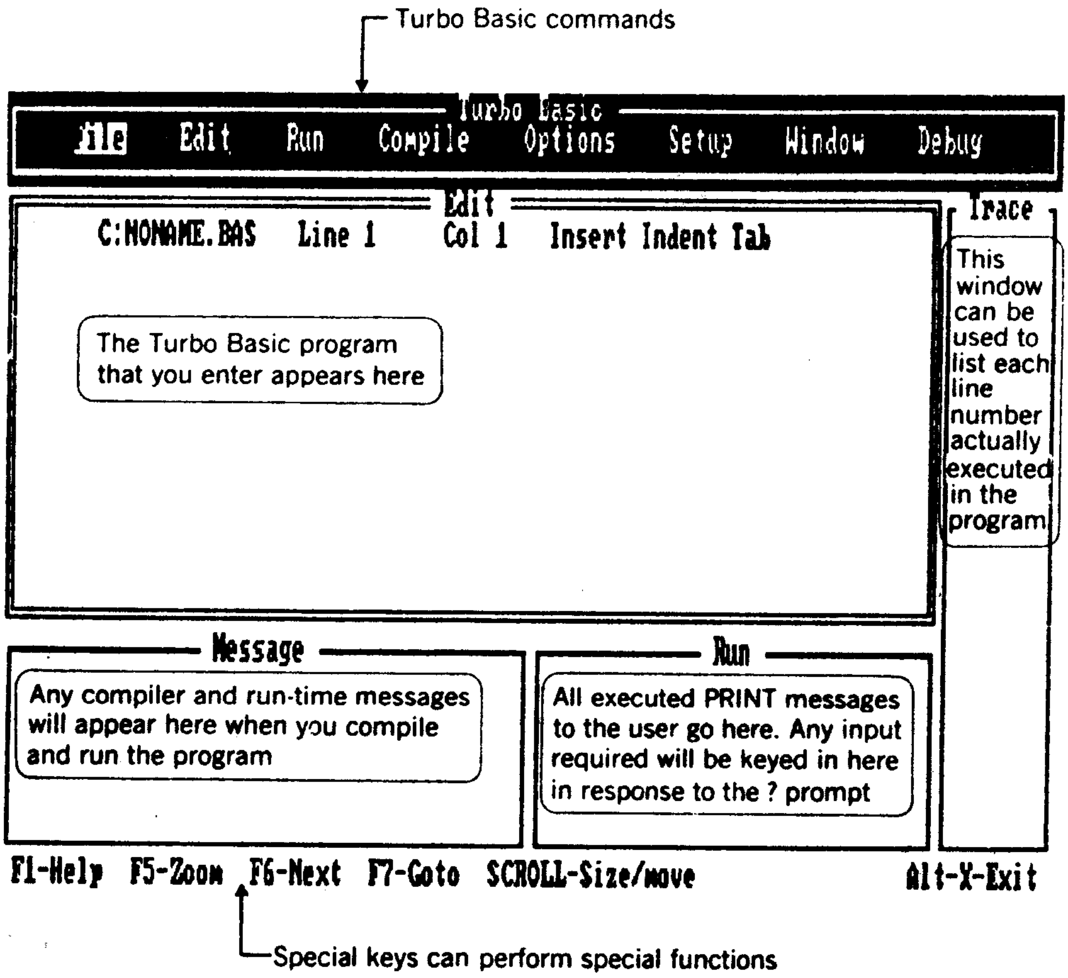


图 2.10 Turbo Basic 主屏幕功能

[10]
[20]
[30] ← 执行该行时，键入第一输入行。
[40]
[50]
[60]
[70]
[80]
} 表示执行 DO WHILE ... LOOP
[50]
[60]
[70]
[80]
} 表示第二次执行 DO WHILE ... LOOP

[90]　　←　输入结尾值: stop, 999, 999 时, 执行该行.

[100]

根据不同 Turbo　Basic 版本, 屏幕上窗口格式稍微有点不同.

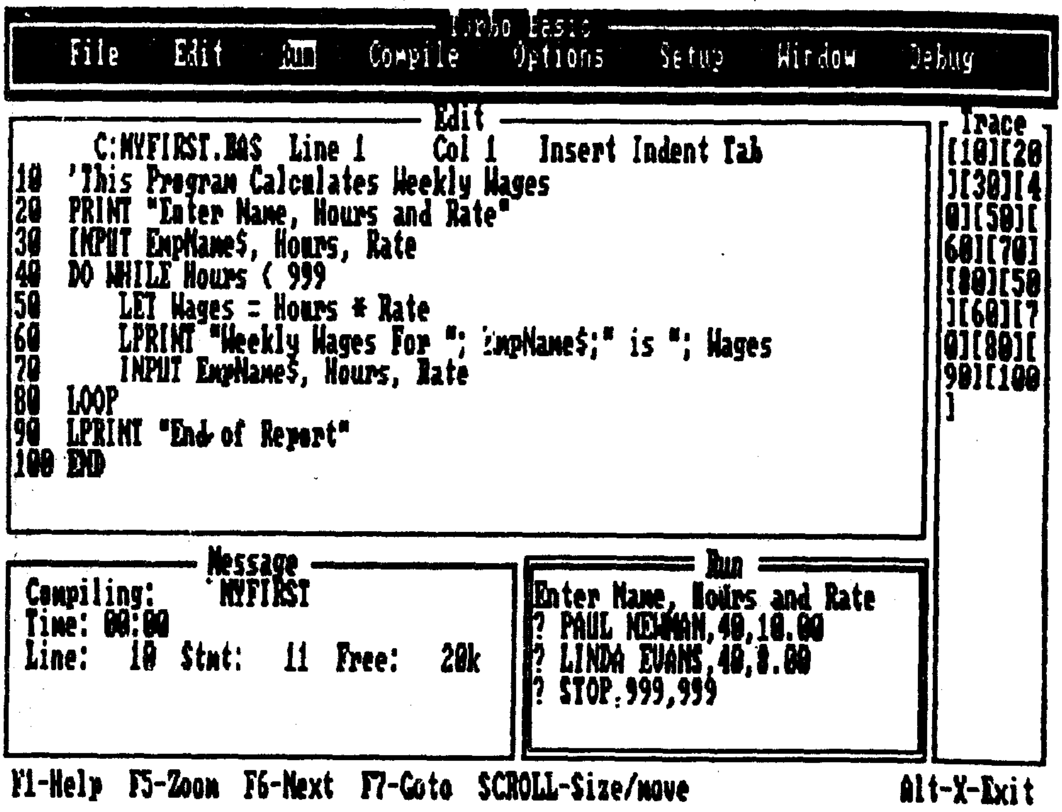


图 2.11　选择 Trace 选项时, 第一章中示例程序的图例

§2.5.2.4　当前窗口

在任何时候, Turbo　Basic 只有一个窗口是"激活"的. 当输入程序时, Edit 窗口是当前窗口; 编译程序时, Message 窗口是当前窗口; 运行程序时, Run 或 Trace 窗口是当前窗口. 用户将很容易确定哪个窗口是当前窗口. 因为当前窗口是双线边界, 而另外一些窗口是单线边界; 例如, 在图 2.11 中打印屏幕时, Run 窗口是当前窗口.

§2.5.2.5　屏幕底部的帮助行

在屏幕底部的帮助行表示如何使用一些特定键. 该行具体显示的键取决于当前所执行的命令。如图 2.10 中 File 命令被高亮度显示, 在屏幕右下侧的 Alt—X 表示: 若同时按下 Alt 和 X 键, 将退出 Turbo　Basic, 回到操作系统. Alt 键在键盘的左下角.

在帮助行中以“F”开头的键表示功能键（Function keys）。在 Turbo Basic 中每一个功能键有不同的含义。

§2.5.3 文件命令和编辑命令

§2.5.3.1 概述

主屏幕顶部的命令有 File，Edit，Run，Compile，Opitions，Setup Window 和 Debug。运行 Turbo Basic 程序时，只用到这些命令中的三个．本节着重讨论这三个命令：File，Edit 和 Run

命令	用　途
File	命名一个程序
	把程序由内存存入磁盘
	从磁盘中调用一个程序
	退出运行
Edit	输入一个新程序或编辑一个现有程序
Run	编译并立即执行 Edit 窗口中的程序．

其它命令将在以后讨论，这些命令指导系统的运行。

为了执行一个系统命令，首先使该命令高亮度显示；用←和→键将光标移到所需命令，然后按回车键。←键在数字小键盘 4 的下部，→键在 6 的下部。

[4 ←]　[6 →]

若是高级键盘，还有一套独立于数字小键盘的光标控制键。

图 2.3 和 2.4 给出也数字小键盘在键盘上的位置。若想从 File 命令移至 Edit 命令，按一次→键．此时 Edit 被加亮显示，按回车便执行 Edit 命令。

为了执行 Edit 命令，那么：

a. 用→键使Edit 命令在主屏上变亮度显示。

b. 按回车键。

File　[Edit]　…　←　按回车就可以编辑一个程序。

总之，为了选择一个命令，我们首先应用光标键使其高亮度显示，然后按回车键；或者，采用别种方法，按要执行命令的第一个字母；例如，在主屏按 E 就执行了编辑命令。这种方法可以省去移光标和按回车。

在讨论 File，Edit，Run 命令前，我们必须给程序取一文件名.

给程序文件取名

每一个 Turbo Basic 程序均是一个文件（File)，用 File 命令，用户可以编写一个新程序，保存和调用已有的文件. 在任何情况下，必须先给程序命名.

文件名的长度为一到八个字符. 对所有程序名， Turbo Basic 自动加一个扩展名 .BAS，所以若命名文件为 MTFIRST，Turbo Basic 将按 MTRIRST.BAS 保存. 这样可以使用户在查阅目录或子目录时立即知道哪个是 Turbo Basic 文件.

程序名可以是大写、小写或大小写混写. MYFIRST 文件与 Myfirst 和 myfirst 是同一个文件.

将一个文件写入磁盘或者从磁盘中调出时，计算机假设该文件是在当前的磁盘驱动器中. 若文件是存入不同的磁盘驱动器或从不同的磁盘驱动器中读出，必须用驱动器字母做为文件名的前缀.

A:myfirst   ← myfirst 程序在 A 驱动器中.

例:

将要存入和调用的文件	含　义
mysecond	文件叫做 mysecond，以 mysecond.bas 的名字存入当前的驱动器中.
C:mythrid	程序名为 mythird，存放在 C 驱动器中.

§ 2.5.3.2 File 命令

Turbo Basic 系统被装入后，主屏幕 File 命令的字符反相，使其高亮度显示，即 File 命令由黑字白底组成，而其它命令是黑底白字.

此时，按回车就可执行 File 命令选项的下拉选单.

此时，File 命令和 Load 选项均为高亮度显示，若按回车，计算机就认为要调用一个已存在的文件.

为了获得 File 命令下拉选单的不同选项，按 ↑ 或 ↓ 光标控制键. 每次按 ↓ 光标键，下拉选单的下一命令就被高亮度显示；要回到原来的选项，可以按 ↑ 光标键.

当某一选项高亮度显示时，按回车即可将它执行.

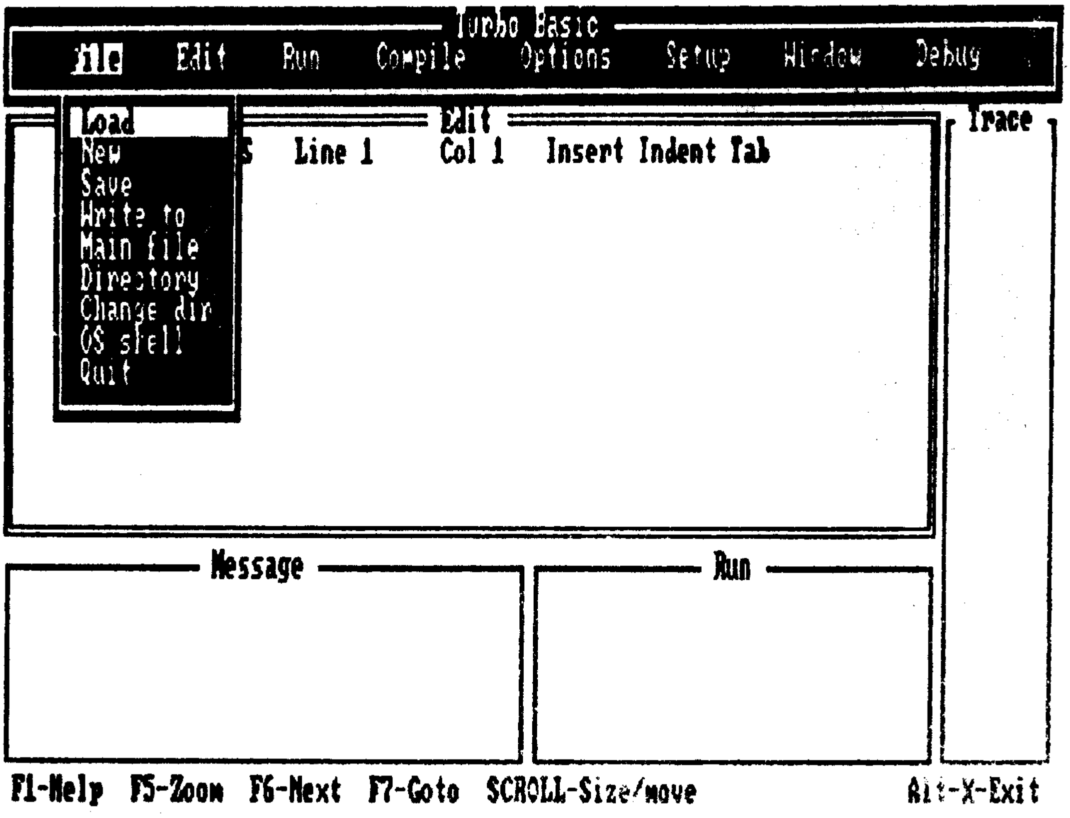


图 2.12 File 命令的下拉选单

下面介绍两个选项:

File 命令下拉选单的选项	
Load	把磁盘中已有的 Turbo Basic 程序调入内存中。
Write to	在输入一个程序之前，进行命名或把一个已有的程序拷贝到一个新文件中。

用 Load 选项把一个已有的程序调入内存中，用 Write to 选项命名一个新的文件。

总之，为了选择一个 File 选项，先用 ↑ 或 ↓ 键使其高亮度显示后按回车。对已讨论过的 Write to 选项，它将询问用户将输入的新程序名。这两种情况都必须等待几秒钟，直至命令执行完毕。

例:

1. 图2.13描绘如何开始建立一个程序文件。

   该选项表示要输入一个新程序并将其存在myfirst 文件名下。该程序被命名为 myfirst.bas，它将存入装有 Turbo Basic 编辑器的当前磁盘驱动器中。

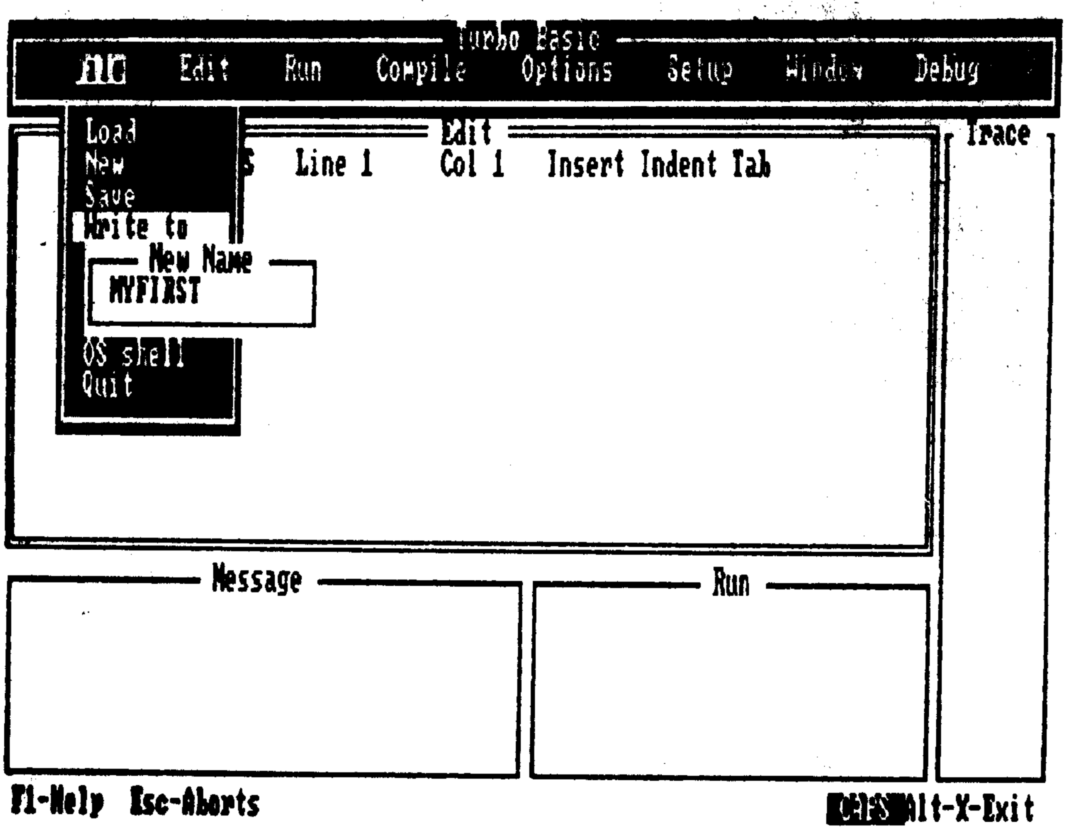


图 2.13　建立名字为 myfirst 的文件

当"Write　to"选项高亮度显示并按回车后，用户将被提示输入一个新文件名。此时键入 myfirst（或 C：myfirst）并按回车。几秒钟后计算机将为这个文件保留空间，然后用 Edit 命令开始输入 myfirst.bas 程序。

2. 图2.14描绘如何将一个已有的程序调入内中。

File 命令下的 NEW 选项：清屏

假如在使用中，要重新开始一个新的程序，可以用 Save 选项保存当前程序，然后选择 NEW 选项，清除编辑屏幕并把原先的程序从内存中删去。然而，New 选项不命名一个程序，所以必须重新进行 File 命令，选"Write　to"以便命名新的文件名。

总之，到此为止，我们已经讨论了四个选项：

选项	功　能
Load	把一个已有的程序调入内存中
New	清屏以便键入新程序

Save	把一个程序存入磁盘中
Write to	命名一个新程序.

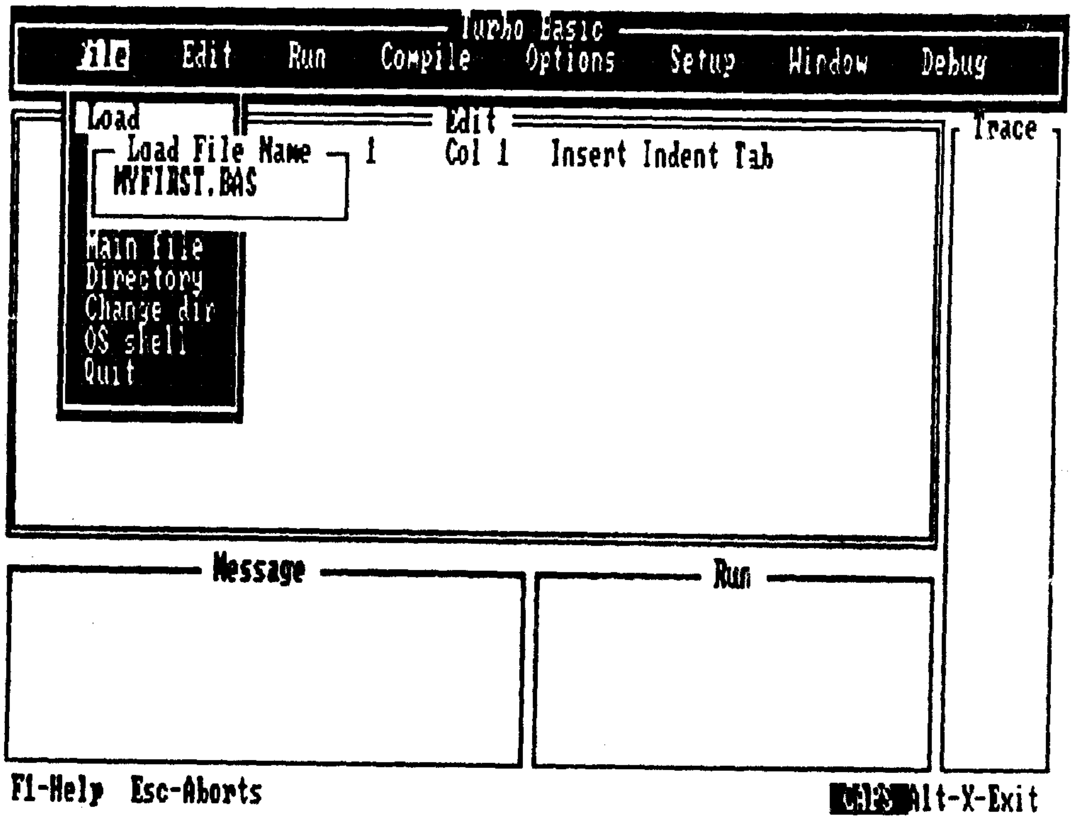


图 2.14 将 myfirst 程序调入内存的方法

本图描述如何将一个已有文件myfirst.bas装入内存中. 由于编译器对大小写不敏感, 所以 MYfirst.Bas 也是可以接受的. .BAS 扩展名可以不写, 因为对于所有 Basic 文件假定有.BAS 扩展名.

§2.5.3.3 Edit 命令

在编辑一个程序之前, 必须从 File 命令转至 Edit 命令, 方法是: (1) 先用光标控制键使 Edit 高亮度显示. 然后按 Enter 或 (2) 敲入 E. 此后, 可以键入一个新的 Turbo Basic 程序或编辑一个已有的程序. 我们说现在处于"编辑状态".

若用 File 命令中的"Write to "选项将一个新程序命名为: myfirst, 那么刚进入编辑状态时, 编辑屏幕的最上面一行, 即所谓的"状态行" (status line)", 显示如下:

```
            C:MYFIRST.BAS   Line 1   Col 1   Insert   Indent   Tab  <-- Status line
Cursor -->  _
```

状态行显示当前的程序名 MYFIRST，以及当前光标所在位置．在上图中，光标位置是第 1 行第 1 列．输入程序程序时，相应位置将发生变化．其它三个条目（Insect，Indent 和 Tab）显示了编辑器的状态．

用 Turbo Basic 编辑器编辑程序时，不仅能够键入指令，而且修改起来，既快又容易．

§2.5.4 使用 Turbo Basic 编辑器输入程序

§2.5.4.1 基本规则

下面的几条基本规则有助于 Turbo Basci 编辑器的使用．

a. 每一行只打印一条指令．
b. 输完一条指令后，按回车将它送入计算机．
c. 键入程序时，以闪烁的下划线形式出现的光标将显现当前的位置，即状态行将显示行号和列号（参见图 2.15)．

§2.5.4.2 修正错误

若输入一行后发现其中有错误，应先用光标键（↓、↑、→、←）将光标移至出错处．按光标键在屏幕上移动光标不会引起正文任何变动．在任何时候，闪烁的光标都将表明当前所处的位置．用下列方法可以对所在行进行变动．

a. BackSpace 键

用 BackSpace 键（←）删除字符．BackSpace 键位于主键盘回车键的正上方，它可以删除光标左边的字符．

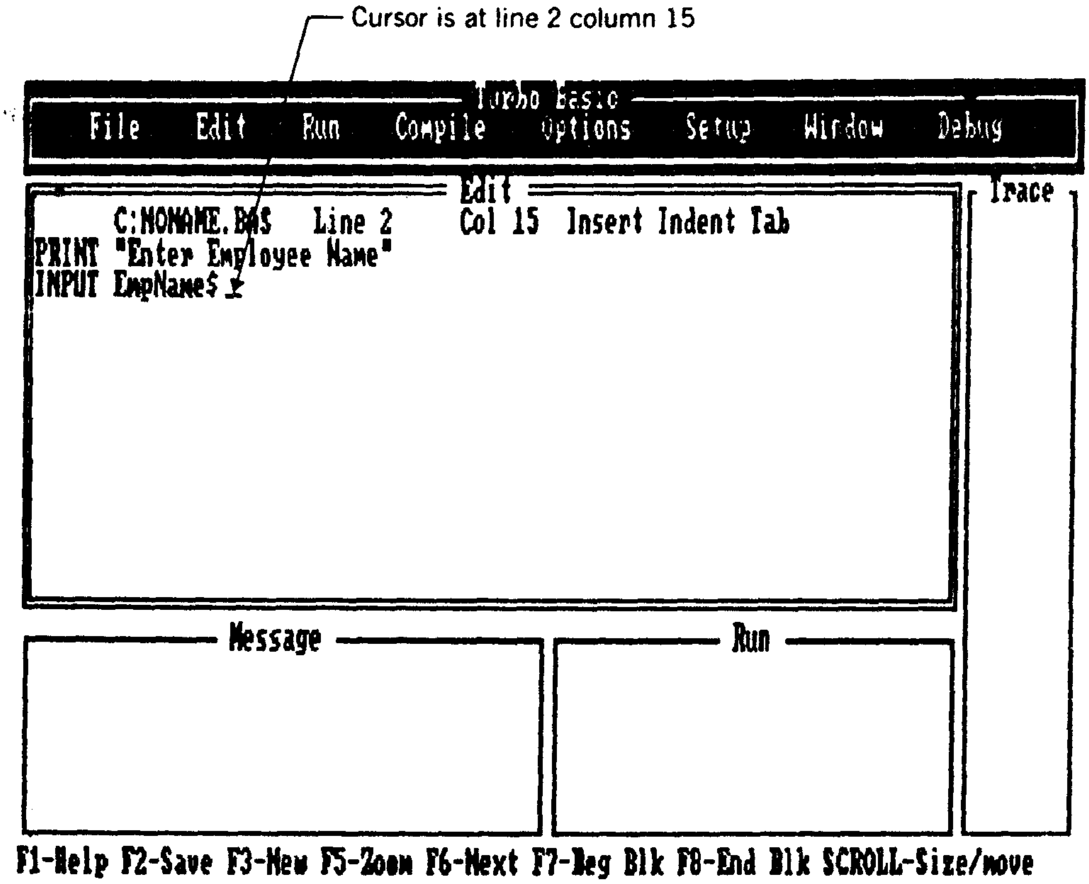


图 2.15 状态行显示当前光标位置

例 1:

打人 Pa 后，发现 a 是错误的．此时，屏幕为:

Pa__ ← 光标位置

按一次 BackSpace 键，将删除“a”，并使光标置于 P 的后面．

P__

然后，用户可以在 P 后键入期望的字母．每按一次 BackSpace 键，光标左面的字符就删除一个，所以不能用该键更正一行开头的错误．

例 2:

PRANT "This is a constant to be printed"

该命令应为 PRINT，而不是 PRANT；若用 BackSpace 键更正，用户将删除所有正

确的字符直到 PRANT 中的 A:

```
PR_
```

这样就需要重新键入 A 后面的每一个字符，所以用 BackSpace 键最好是在不正确字符一键入就更正。例如，若要键入 PRN，最好此时安 BackSpace 删除 N。用户也可以用 BackSpace 键更正前一行最后或接近最后的错误。

例 3:

```
PRINT   "Enter Employee Name?
INPUT   EmployeeName$_       ← 此时，发现前一行最后为 "，而不是 ?
```

那么，使用↑键把光标移到前一行，然后用→键把光标移到 ?的后面一列；此时，屏幕为:

```
PRINT   "Enter employee Name?_      ← 光标位置
INPUT   EmployeeName$
```

然后，按 BackSpace 键删除 ?，最后键入引号。

b. Del 键

用户也可用←和→以及 Del 或 Delete 键删除某个字符。BackSpace 键删除光标左边的字符，而 Delete 键则删除光标所在处的字符。

```
PRINT  "The Mercedes 569 SL is sprotycair"_    ← 此时，发现 car 拼写错了
```

按←键三次使光标移到 i:

```
Cair     ←  如果用户按 Delete 键，则变为:
Car
```

这就是说，Delete 键删除光标所在的字符。

c. 插入字符

为了插入一个字符，将光标移至想要插入的地方，然后插入所需的字符。

例 4:

```
PINT  "This is it!" _      ' "PINT"应为"PRINT"
```

用 BackSpace 键回到 PINT 的 I 处，然后加入 R；这样就使 I 后面的所有正确字符也删掉了。比较好的方法是：用←键把光标移至一行开头，作出改变:

1. PINT "This is it!" _　←　光标所在处
2. 按←键17次，光标位于：　PINT "This　is　it!"
3. 键入R，R将会插入到I前面，出现PRINT。

d. 用一个字符替换另一个字符

例5:

cat　←　按"O"，结果为"Coat"。

状态行中的Insert表示字符插入光标所在处而不是替换已有的字符。为了替换已有的字符，需先按一下键盘右边的Ins或Instert键。假如要输入Cat而不是Cot:

Cot　←　先按一次Ins键，后键入a，这样cat就代替了cot。

按Ins或Insert键时，编辑屏幕的状态行中不再出现Insert；这就意味着，现在不处于"插入"状态，而是"替换"状态。

Ins或Inser键是一个开关键。即按一次，进入替换状态（此时单词"Insert"在状态行中消失）；再按一次Ins键，又进入插入状态（Insert重新出现在状态行中）。该键的功能象一个"乒乓开关"。

总之，可以用←和→键将光标移至一行的任何地方而不发生任何变化。若要加一个字符，把光标移到想插入处的右边，再键入所需的字符；若要替换一个字，按一次Ins键切换到替换状态，先将光标移到想替换的字符下，然后按所需的字符。

e. 为了另起一行，应在该行的最后按回车键

为了在更正前一行后开始新的一行，把光标移到前一行最后或用END键，然后按回车键。

例6:

```
PRINT  "Employee Nmaf"
INPUT  Employee Name $ _
```
←　此时发现上行"Namf"拼写错误。

修　正　步　骤

把光标移到"f"
按Ins或Insert键切换到替换状态
键入字符e替换f
将光标移到INPUT行的末尾
按回车后，继续输入程序

f. 若要删除一行，用↑和↓把光标移至要删除行，同时按Ctrl 和Y键，便将该行删除。

例 7:

```
PRINT  "Enter"
PRINT  "Enter Employee Name $"
INPUT  EmployeeName $_     ←  此时，发现第一行不必要
```

修 正 步 骤
把光标移到第一行
按 Ctrl—Y
把光标移到 INPUT 行最后，按回车后继续输入程序.

g. 若要在某行前加一行，把光标移至该行前面按回车，计算机将插入一空行，并把光标移至空行。

例 8:

```
PRINT  "Now is the time for all good man "
PRINT  "of their country"
```

程序行: PRINT "to come to the aid" 被不小心忽略掉. 为了把这一行加入，把光标称到第二个 PRINT 的 P 处，按回车键. 此时，在两个 PRINT 间有一空行，按一次↑把光标上移，再输入被省去的行.

```
PRINT  "Now is the time for all good man"
PRINT  "to come to the aid "
PRINT  "of their country"
```

h. Home键可以把光标直接移至一行最前面，而End 键则把光标移到最后、

§ 2.5.4.3 缩进编辑

为了缩进编排，可以使用键盘左上方的 Tab 键 (⇄). 每按一次这个键, 光标将移动 n 个空格. 将逻辑控制结构缩排，以便于整个结构易于辨认:

```
DO  WHILE
     LET  ...
     INPUT            } 这几行被缩进
       :
LOOP
```

几乎所有的编辑器均有上面的编辑功能，这对于输入简单的 Turbo Basic 程序已经足够了．然而 Turbo Basic 编辑器是非常成熟的，与其它的相比有更多的功能和选项．Turbo Basic 编辑器所用的非常全面的 Wordstar 命令结构．第九章将介绍其它一些非常有用的文本编辑功能和 Turbo Basic 特性．

一旦在编辑状态下输入整个程序后，应将它存入磁盘．在该状态下，只需按功能键 F2．注意，以编辑状态下，帮助行有:

F2 — Save

从图 2.15 中可看到，屏幕底部的帮助行表示在编辑状态 F2 的功能用于存贮一个程序．在运行程序，调用别的程序和开始新的程序之前,或结束上机上前，必须进行存盘，有些时候，若忘记存盘,用户将得到要求存盘提示．

对于一些版本的 Turbo Basic，在运行之前必须选择 File 命令的 Save 选项，如果忽略了存盘，Turbo Basic 会询问是否需要存盘，若要只需键入 Y．

§2.5.5 键盘上的特殊功能键

现在讨论两个在 Turbo Basic 中有特殊意义的键．

§2.5.5.1 提供上下文帮助的 F1 功能键

同许多拥有用户满意接口的程序一样．无论在什么时候，有疑问都可以简单地通过使用 F1 功能键获得帮助．这个帮助功能提供了当前屏幕上高亮度区命令选项的一览表．因为此显示与当前所处的位置有关，所以我们称之为"上下文有关帮助 (Context-Sensitioe help)"．在 File 选项使用 F1 功能键与编辑状态中使用 F1 功能键会得到不同的帮助显示，阅读完帮助显示后，为了删除帮助屏，按 Esc 或者 Escape 键．

§2.5.5.2 Esc （或 Escape）键

Esc 键不仅具有使计算机从 Help 选单中退出的功能，而且可以使计算机从任何一个当前显示命令中退出．从而使计算机可以开始执行另一个命令．在使用 Esc 键退出编辑命令之前，应使用 F2 功能键将用户程序存盘．

避免常见错误

1. 在输入程序之前，一定要退出File命令并进入编辑命令。Write to （输入）文件名后，计算机仍处于 File 命令中。若忘记退回到编辑状态，可能就要出现问题。

   例如，在"Write to "选项下，先键入MYFRIST，接着键入程序。若第一条指令的字母字母是 L，则计算机认为要装入一个现有的文件。这是由于当前处于 File 命令中，从而产生了错误。若第一个指令的字首是字母 O，则计算机将退回操作系统或进入"Os shell" （这是 File 命令中的一个选项）。为了避免出现类似的常见错误，请在键入文件名后，使用 Esc 键，并用光标移动键使 Edit 命令加亮，然后按回车键。

2. 如已经键入，调试和运行过文件MYF1RST，并准备开始另一个新程序。但忘记按 F2 功能键在编辑模式将 MYF1RST 存盘，在选择 File 命令中新的选项之前，可以使用文件命令中的"Save"选项，将程序 MYF1RST 永久地储存在磁盘上。也可以重新回到编辑状态，用 F2 功能键将文件存盘，若这两种方法都没有被采用，将得到如下提示：

   ```
   MYFIRST not saved. Save? (Y/N)
   ```

   此时键入 Y 键以确保 MYFIRST 已存盘。

§2.5.6 Run 运行命令

在编辑状态中，使用 F2 功能键将程序以 MYFIRST 的程序名存放在磁盘上之后，可以用"Compile"命令进行编译或运行。Run 命令包括两个过程：一个是编译，另一个是执行此程序。至此，当程序输入到计算机中并已存储好之后，将利用与该命令中的"Run"去编译和执行程序。

程序输入并使用 F2 功能键存盘后，用 Esc 键退出编辑状态，然后使用光标键使"Run"命令加亮，按回车键。或者，用按字母 R 的方式来选择"Run"命令，以此来替代以上的操作。

选择了"Run"命令后，程序将进行第一次编译。在编译过程中，若出现语法错误，计算机会自动地回到编辑状态，而且光标将指出出现错误的地方。这使我们改错工作十分简便。

使用 Turbo Basic 的一个便利之处就在于：当发现语法错误时，计算机马上转到编辑状态，从而使我们不必从 Run 命令转到 Edit 命令。而是立即就可以改正程序中的错误。在此改错同我们在前面讨论过的文本编辑方式完全一样。改错完成后，按 F2 功能键将改正后的程序存盘。用 Esc 键退出编辑状态，然后再次进入 Run 命令重新编译和执行程序。

程序编译完成后，在 Run 屏幕上将显示出输入和输出。例如：若 INPUT Amt2 是第一个执行指令，提示“ ? ”将出现在 Run 窗口，这意味着计算机正在等待输入。

同先前说明的那样，在提示后键入输入并回车。将输入传送给计算机，则计算机将在下一行显示出输出信息。如果在输入过程中出现失误，可以同通常一样使用 BackSpace 键。

因程序中出现逻辑错误，要进一步修改时，需用 Esc 键退出运行状态；然后使用 Edit 命令高亮度显示，按回车键进入编辑状态，再进行修改。切记，将修改后的文件须用 F2 功能键存盘后再运行程序。由于已将程序储存，用户可以在任何时候中断与计算机的对话。在任何其它时候，要运行此程序时，可以先键入 TB，然后按回车键将 Turbo Basic 的编辑器和编译器装入，接着进入 File 命令的 Load 选项。计算机询问文件名时，键入 MYFIRST。

```
FILE
 LOAD
+--------------------+
|   Load File Name   |
|       *.BAS        |
+--------------------+
```

← 在此处键入 MYFIRST，替代 *.BAS，然后按回车键（扩展名 BAS 可有可无，Turbo Basic 会自动插入）。

为了鼓励读者试着掌握 Turbo Basic 的编辑器和编译器，在本章的结尾给出了几个程序，这中间有正确的也有错误的。

§ 2.5.7 Options 命令

到目前为止我们重点讲述了 File，Edit 和 Run 命令。利用这些命令可以键入、编译、储存和运行简单的 Turbo Basic 程序。其它的系统命令能给编写和调试程序带来很大的灵活性，但并非一定要掌握。

可是，Options 命令具有一个对初学的程序员十分有用的功能。这就是允许用户在任何希望中断程序运行的时刻中断程序的运行。为了理解这个功能，让我们考虑下面的例子。

带有逻辑错误的 Turbo Basic 程序

```
10  INPUT  Amt1
20  DO  WHILE  Amt1  <>  0
30        PRINT  Amt1
40  LOOP
```

```
50   INPUT  Amt1
60   END
```

这个程序是不合逻辑的，它给了一个意义含糊的初始输入 Amt1，这是因为错误地颠倒了 40 行和 50 行的次序。就是说程序输入 Amt1 并且在 DO WHLTE ... LOOP 中重复打印同一个 Amt1。为了使此程序意义明确，就该将 INPUT Amt1 做为 DO WHILE ... LOOP 中的最后一个指令。按这种方式，此 LOOP 打印和输入大批的数量。DO WHILE 将重复处理程序当前所读的初始值。

如果按这个带有逻辑错误的程序输入并运行，则将会出现死循环(infinite loop)。一次又一次地打印同样的 Amt1，除非使用能够使用户终止程序执行的 Turbo Basic 选项。否则，就要关掉计算机，然后再重新起动（注：有两种方式重新起动或重新引导计算机系统：(1) 关掉计算机，并等待 45 秒钟，重新开机；(2) 使用组合键 Ctrl–Alt–Delete，即在按 Ctrl 键的同时，按下 Alt 键和 Delete 键）！这意味着初始化编译器重新装入操作系统，然后通过键入 TB 装入 Turbo Basic 系统，接下来再次装入用户 Turbo Basic 程序，这既浪费时间，又烦琐。

Turbo Basic 提供了一个选项，它可以在程序运行中任一时刻停止程序的运行，甚至包括死循环过程。为了防止在编程中出现前面提到的逻辑错误，或者在任一时刻终止程序运行，建议立即掌握这个选择。

在主屏幕上的第五个命令称作 Options 命令。利用→光标移动键将高亮度区移至 Options 命令，并按回车键，则进入 Options 命令，显示图 2.16 所示的下拉选单。

下拉选单的第三行显示为：

```
Keyboard  Break  OFF
```

需要将其改为：

```
Keyboard  Break  ON
```

为此，先按两次↓光标移动键，将高亮度区移至 Keyboard Break 选项选择 Keyboard Break 条目，然后按回车键。这时，OFF 将改为 ON，屏幕显示成：

```
Keyboard  Break  ON
```

该选择可以通过在按 Ctrl 键的同时按 Break 键（记作：Ctrl–Break）的方法，退出正在执行的程序。在运行状态下，当同时按下 Ctrl 键和 Break 键程序运行被终止，即使是死循环也如此（在标准键盘上，Break 键等同于 Scroll Lock 键，而在 Enhanced 键盘上等同于 Pause 键。）

Keyboard Break ON 处于高亮度区后，可使用 Esc 键退出 Options 命令。

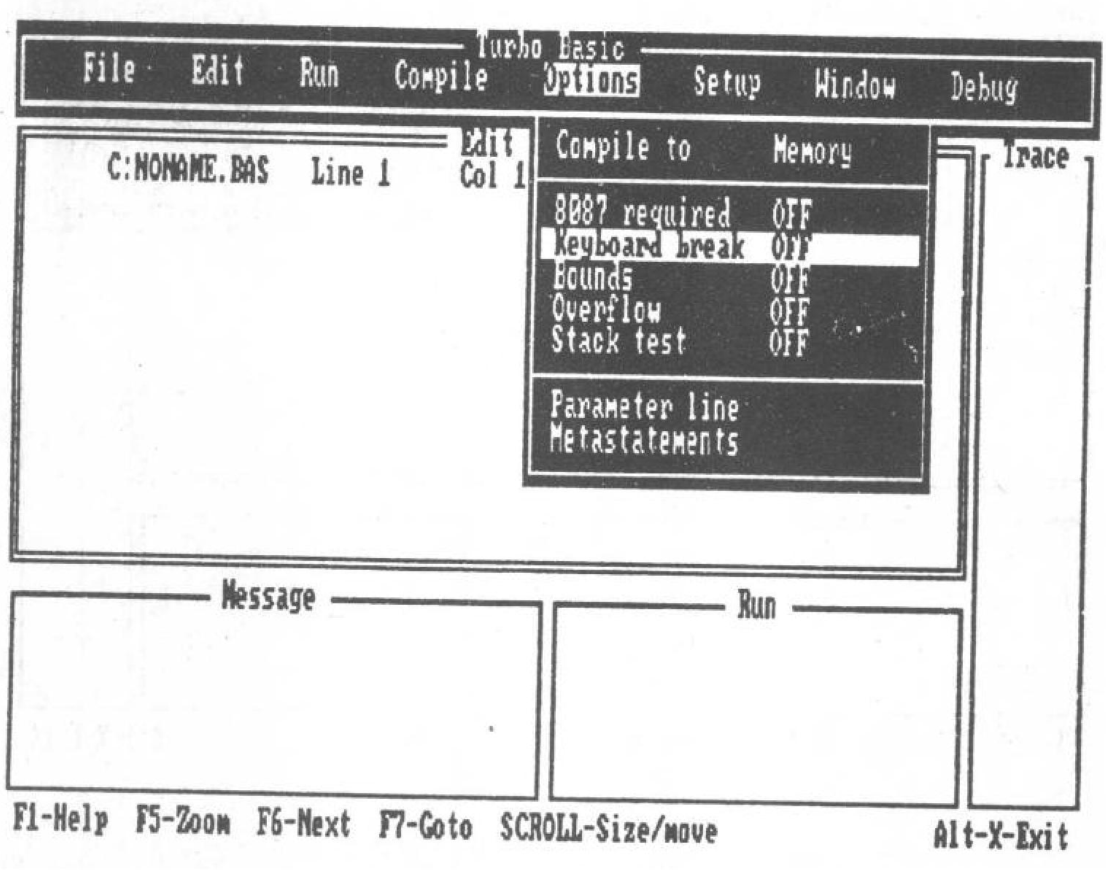


图 2.16　Option 命令的下拉选单

总之，在运行状态中，为了在任何时候退出已给定的程序，可：

1. 使用→光标移动键选择Option命令，然后回车，或按字母O。
2. 使用↓光标移动键两次，使Keyboard Break OFF 加亮。
3. 按回车键，从而显示出Keyboard Break On
4. 使用Esc 键退出Option 命令。
5. 最好将Keyboard Break ON 储存起来，使其成为标准功能或引导时系统的设置。为此，可以按以下方式操作：按→光标键使 Setup 命令加亮，并按回车键；或者，按字母 S 进入 Setup 命令。
6. 在Setup 状态中，按↓光标键4 次，使高亮度区在Setup命令选单的SaveOptions／Window／Setup 上(参见图 2.17)。

按回车键，将选定的 Keyboard Break ON 选择保存起来。计算机将指定系统设置文件名为 TBCONFIG.TB，并用该文件将以上作出的改变保存(参见图 2.18)。

若按回车键，计算机将要求用户确认：是否将这个选择储存?(参见图 2.19)。

键入 Y，表示同意。至此，前述的六个步骤将永久性地保证：可以通过使用组合键 Ctrl—Break 中止程序的执行。它将为所有 Turbo　Basic 程序的编译和执行服务。

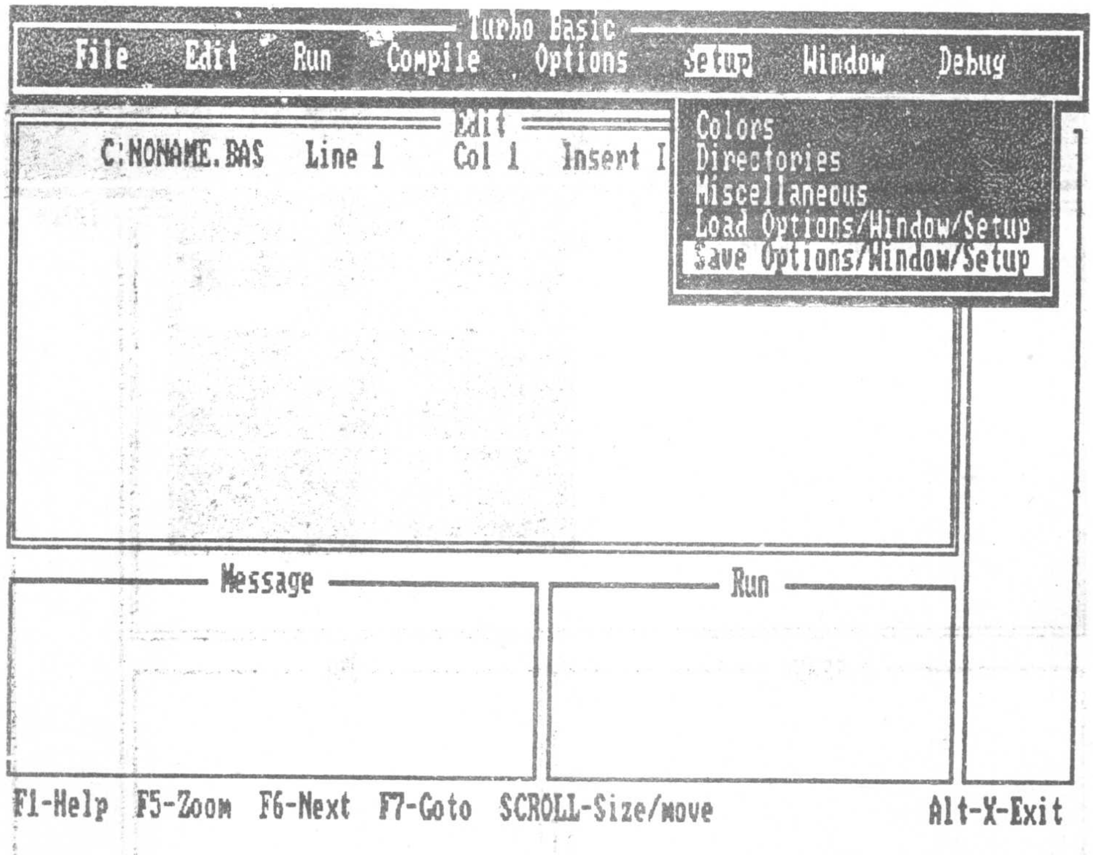


图 2.17　Setup 命令的下拉选单

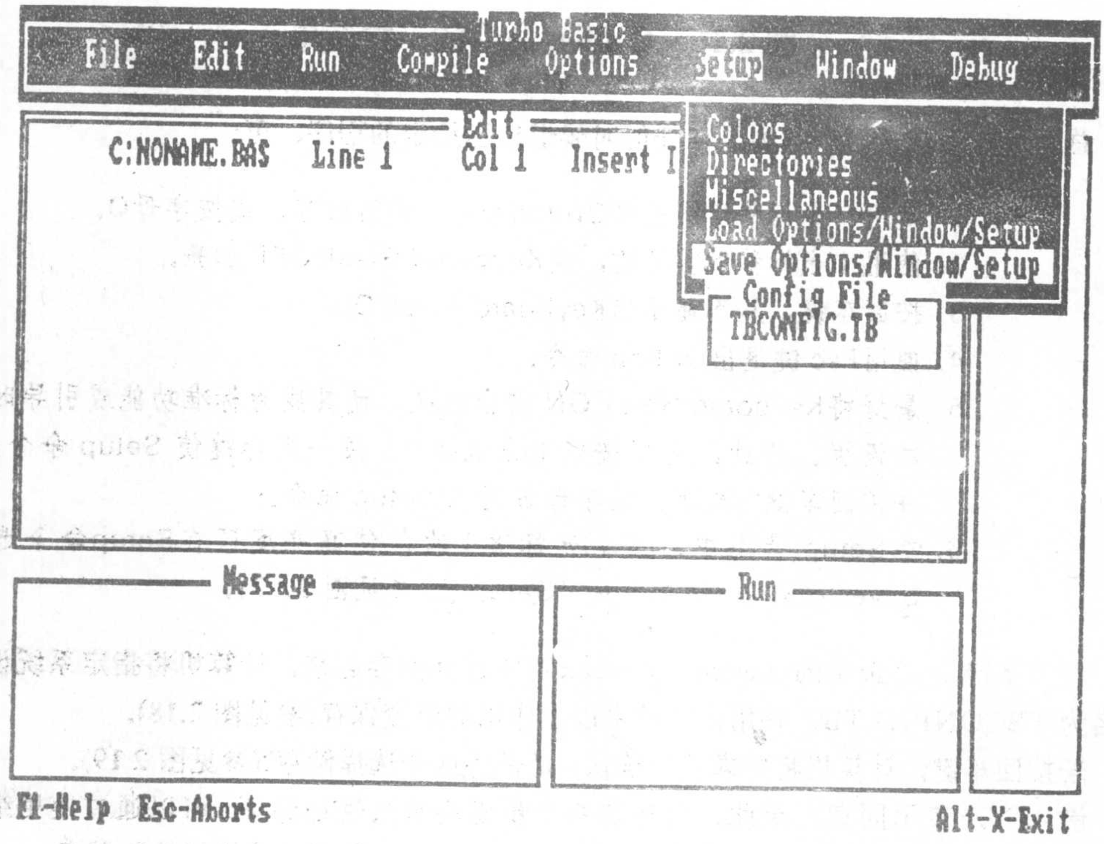


图 2.18　TBCONFIG.TB 文件储存对 Setup 作出的改变

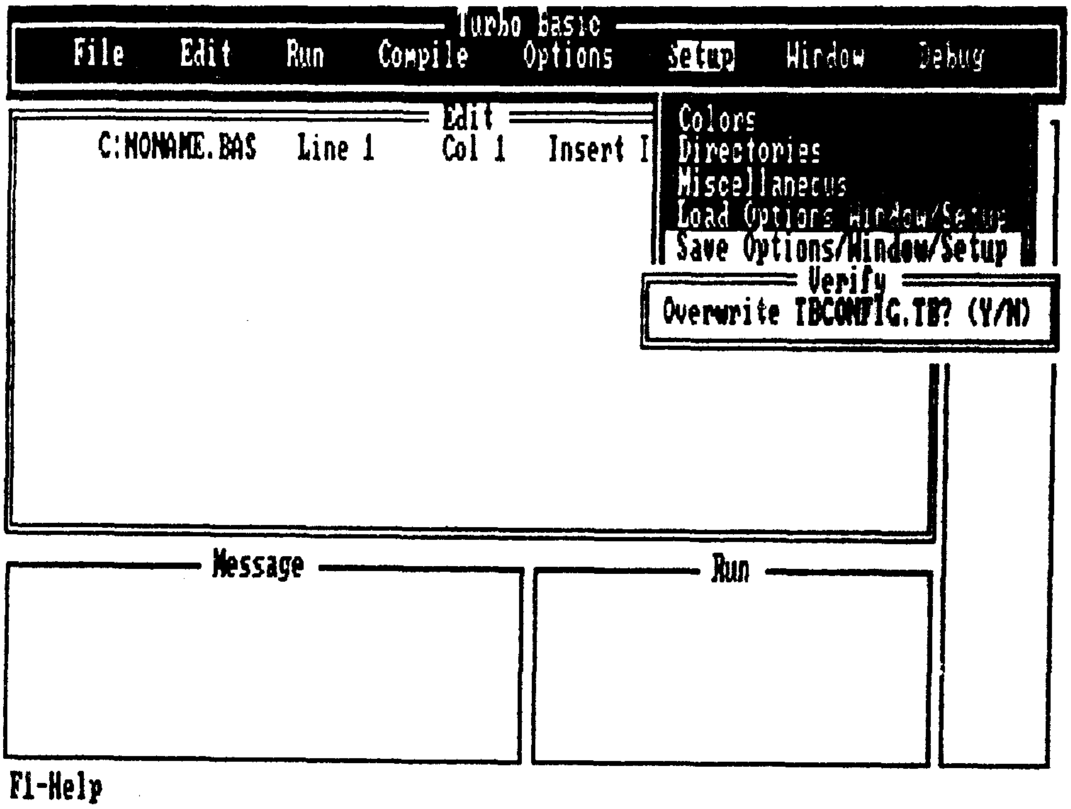


图 2.19 TBCONF1G.TB 文件存盘前，计算机要求确认

我们极力推荐使用前述六个步骤永久性地形成系统配置:

Keyboard Break On

其它的选择和命令将在后面部分介绍。下面给出一个简单的命令小结:

命 令 小 结	
File	装入、储存或定义一个程序文件。
Edit	输入和修改程序文件。
Run	执行程序，若程序没翻译首先要编译。
Compilc	编译程序。
Options	建立一定的技术要求，如 Keyboard Break 选择。
Sctup	使用户能够控制 Turbo Basic 环境。
Window	配置窗口以适合个人的需要和爱好。
Debug	帮助调试程序。

特定功能键小结

功能键 F1	显示上下文帮助
功能键 F2	在编辑状态，将程序存盘.
Shift-Prtsc(或 PrintScreen)	回送打印当前屏幕显示.
Ctrl-Prtsc (或 PrintScreen)	回送打印将要出现的屏幕显示
CapsLock	输入大写字母（按一次输入大写，第二次按输入小写）.
NumLock	用数字小键盘输入数字，取代光标移动键（按一次为输入数字，第二次按成为光标移动键）.
Shift (↑)	输入大写字母，或输入基本键盘上数字上方的特殊符号.
BackSpace(←)	光标退一格，并删除光标左侧的字符.
Enter	将输入传送给计算机.
Ctrl-Alt-Delete	重新引导或重新起动计算机.
Tab (⇌)	缩排.
INS (或 Insert)	在编辑状态中将插入状态改为替换状态，（按一次为替换状态，再按一次为插入状态）.
Del (或 Delete)	删除光标处的字符.
Ctrl-Break	终止程序运行.
光标键 (↓ ↑ ← →)	在编辑状态下，在屏幕上移动光标在主选单上选择不同命令.
Home	在编辑状态下，使光标返回到一行的开头.
End	在编辑状态下，使光标返回到一行的结尾.
Ctrl-Y	在编辑状态下，删除光标所在的一行.

§ 2.6 本章辅导

§ 2.6.1 本章小结

输入和运行 Turbo Basic 程序的方法

1. 用操作系统引导系统，在正常情况下显示出日期和时间提示。
2. 将Turbo Basic 系统安装在计算机上，然后在提示符（A>或C> 下键入TB，装入 Turbo Basic 编辑器和编译器。若 Turbo Basic 系统在子目录中，必须在键入 TB 之前，用指令："CD＼<子目录名>" 进入子目录。
3. 初始屏幕显示后，按回车键进入主屏幕。
4. 将File命令加亮，按回车键，文File命令的选项将由一个下拉选单给出。
5. 按↓光标移动键三次，使Write to 加亮，按回车键，进入此选项.
6. 计算机将提示：程序文件名. New Name:
7. 键入程序名，等待命令执行，然后按Ese键退出此命令。
8. 按→光标移动键，将Edit 命令加亮，按回车键。
9. 进入编辑状态，可以输入用户程序。
10. 当Turbo Basic程序输入完成后，按F2功能键将程序存盘，然后用Esc 键退出编辑命令。
11. 按→光标移动键，将Run命令加亮，按回车键，程序进行第一次编译，若存在错误，错误将在 Edit 窗口显示出来，光标回到出现错误的那一行。系统自动回到编辑状态，使错误很快被查出，并加以改正，然后用 F2 功能键存盘。重复执行此过程直到程序没有语法错误。
12. 没有语法错误后，Run窗口显示下提示符?提示输入，输入测试数据后，一般情况 Run 窗口将以打印语句的形式显示输出，检查输出是否正确，若不正确，按 Esc 键，并用←光标移动键返回编辑状态，检查程序，重新执行10–12。
13. 程序经充分测试输出，结果正确。若需要，用shift–prtse(或Print Screen) 打印所有的四个窗口。
14. 当完成以上步骤后，关掉计算机，若想回到操作系统，使用Alt– X 键，退出 Turbo Basic。

§ 2.6.2 本章自测

1. IBM微计算机分为两大类，IBM ________系列和IBM ________系列。
2. 磁盘有两种尺寸；________英寸和________英寸。
3. 英寸的磁盘通常用于IBM PS / 2 计算机。
4. （T或F）硬盘的数据储存量远多于磁盘。

5. 硬盘驱动器通常称为________驱动器。
6. （T或F）起动计算机时，操作系统一定装入在计算机中。
7. 操作系统的第一个提示为________，随后的提示为________。
8. 回答操作系统后，计算机显示为A>，则称此为________，并且系统设定或记录驱动器为________驱动器。
9. 为了拷贝A驱动器中的Turbo Basic磁盘到硬盘，可以利用________指令。
10. Turbo Basic主屏幕最上一行表示________。
11. 为将一个命令状态转移到另一个状态，或________或________。
12. 在Edit命令状态，当前窗口为________窗口，当前窗口被记为________。
13. 开始程序编码称为Tesitit，其执行那些步骤。
14. Testit输入后，键入________存盘。
15. 在任一屏幕下，按________键以显示Help屏幕。
16. 如果已选择了________命令中的________选项，将可以永久利用________在Run状态中终止程序运行。
17. 输入程序时，假如按了字母i而不是a，并直到已输三行时才意识到出现错误，指出如何改正错误。
18. 设已输入

```
PRINT "Good morning"
INPUT Employeename $
```

请指出如何在这两行之间加上：PRINT "Employee Name"。
19. 若在键入PRINT时，错误地键入了PRIST，如何改正错误？
20. 若在键入INPUT时，错误地键入了INPT，如何改正错误？
21. 什么是"乒乓"开关键？
22. (T或F)在没有执行过Compile命令的情况下执行Run命令，程序当然不会运行。
23. （T或F）为了阅读方便，Turbo Basic允许在DO WHILE ... LOOP中出现缩排行。
24. 为了退出编辑状态，并执行程序，应首先按________，然后按________键，并且按R键使程序运行。
25. （T或F）若程序中出现语法错误，则错误将显示出来，而且计算机自动地返回编辑状态，光标指示出错误，以便于改错

§2.6.3 实例

输入营业员姓名，营业额，利润（用小数形式0.06表示6%利润）；并按如下形式在打印机上打印：

营业员姓名　营业额　全部利润

全部利润 ＝ 全部营业额×利润率。

RUN 窗口

?Enter Name，Amount of sale，commission

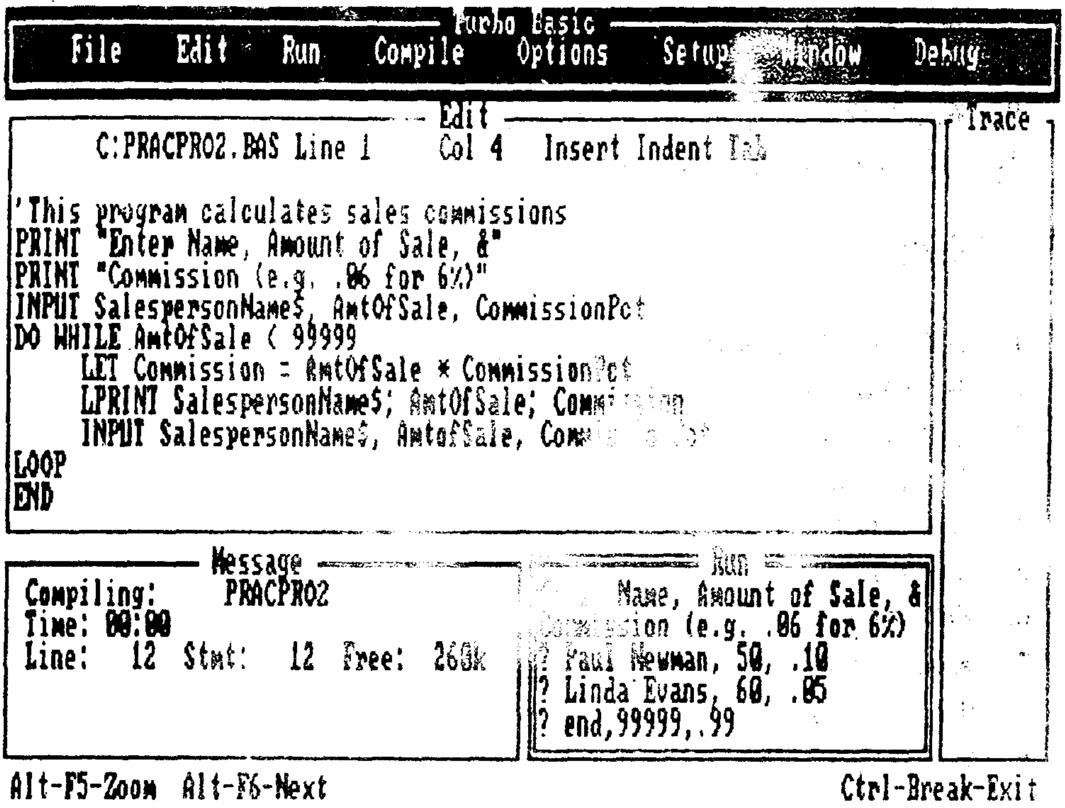
Paul Newman，10

输出结果

Paul Newman 50.00 500

营业员名字的终值是 end、营业额的终值是 99999，利润的终值是 0.99。虽然实际上只有利润的终值 99999 在 DO WHILE ... LOOP 中被检查，但所有三个变量必须全部键入。

程序的输入，输出见图 2.20：



(*a*)

图 2.20(a) 实例的程序

Paul Newman 50 5

Linda Evans 60 3

图 2.20(b) 实例输出结果的硬拷贝

§2.6.4 复习题

Ⅰ. 是非题

1. Turbo Basic是一种需要编译的符号语言（而不是解释程序）
2. Turbo Basic 程序的每一行都被尽可能地编为一个代码.
3. File 12 是一个非法文件名.
4. Turbo Basic 是一个操作系统程序.
5. 文件名必须由8个字符或少于8个字符组成，并且有一个由三个字符组成的扩展名.
6. Turbo Basic程序不要求行号
7. 当刚刚完成一个程序，准备开始编一个新的Turbo Basic 程序，应该将高亮度区置于 File / Write to 的位置.
8. 硬盘储存的信息多于磁盘.
9. 在运行程序前,应拷贝一份工作备份盘.
10. 计算机可以由组合键Ctrl–Alt–shif重新起动.

Ⅱ 综合题

1. 在Turbo Basic屏幕的Edit窗口中显示出的是什么?
2. 在Turbo Basic 屏幕的Run窗口中显示出的是什么?
3. 已进入文件命令中的Write to 选项，这时使用Help功能，屏幕上将显示出什么样的帮助信息.
4. 确定在Edit状态中使用F5功能键的含意.
5. 指出在Edit状态中使用Del（Delete）键将会出现什么结果.
6. 指出在Edit状态中Ins键的含意.
7. 装入程序Ball; Ball 为Turbo Basic 的一个示范程序（其在Turbo Basic盘中)，指出 Ball 执行了什么功能.
8. 装入并执行Turbo Basic 程序盘中的程序Hanoi.
9. 指出文件存盘的两种方法.
10. 给一个程序命名时有什么规则.

§2.6.5 调试练习

输入下列程序，修改需要改正的错误，然后运行，使其能正常工作.

```
. . .
PRRIN Enter Name, Hours and Rate"
INPUT EmpName, Hours, Rate
DO  WHILE Hours > 999
    LET Wages ? Hours * Rate
    LPRIT "Weekly Wages Fo ";Empname$;" "is"; Wages
LOOP
INPUT EmpName$, Hours, Rat
LPRINT "End of Report
END
```

§2.6.6 编程题

调试下列程序，它能帮助用户学习如何输入，编译和运行一个程序.
为了执行程序，可以使用其它资料中提供的测试数据，也可以自己设计.

1. 输入图2.21 给出的程序. 并使用用户自己设计的数据运行.
2. 输入图2.20(a)给出的实例，并使用用户自己设计的数据运行.
3. 图2.21是一个Turbo Basic 程序的具体示例.
   (a) 描述输入
   (b) 描述输出
   (c) 用自己的话，叙述将输入转经为输出的过程.
   (d) 输入用户自己设计的数据运行此程序.
   (e) 在程序前写出伪代码.

```
'This program calculates profit
PRINT "Enter Unit Price, and Cost"
INPUT UnitPrice, Cost
DO  WHILE UnitPrice <> 0
    LET Profit = UnitPrice - Cost
    LPRINT "Profit = ";Profit
    PRINT "Enter Unit Price, and Cost"
    INPUT UnitPrice, Cost
LOOP
LPRINT "End of Report"
END
```

图 2.21 编程题 3: 计算效益

4. 图2.22是一个Turbo Basic 程序的具体示例.
   (a) 描述输入
   (b) 描述输出
   (c) 用自己的语言，叙述将输入转化为输出的过程.
   (d) 输入自己设计的数据运行此程序.

(e) 在程序前写出伪代码

```
' This program inputs exam grades and computes an average for
' each student
INPUT StudName$, Exam1, Exam2, Exam3, Exam4
DO WHILE Exam1 < 999
        LET Total = Exam1 + Exam2 + Exam3 + Exam4
        LET Average = Total / 4
        LPRINT "Average for "; StudName$; " is "; Average
        INPUT StudName$, Exam1, Exam2, Exam3, Exam4
LOOP
LPRINT "End of Run"
END
```

图 2.22 编程题 4: 计算学生的平均等级

第三章　输入／处理／输出：编写基本程序

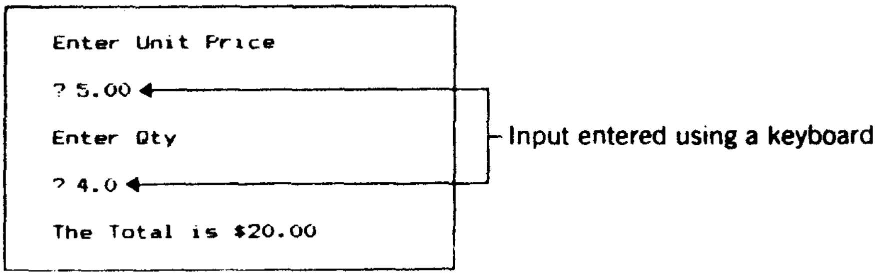
本章要求

熟悉如下内容：

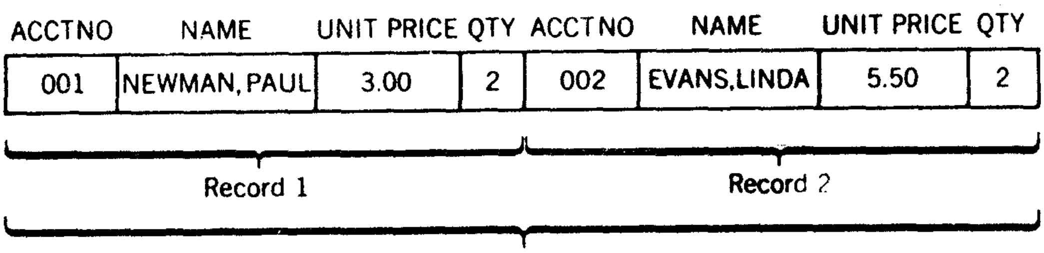
1. 程序的基本结构；
2. 变量和常量；
3. 如何输入；
4. INPUT、LET、PRINT、LPRINT指令的格式；
5. 如何完整地编写基本Turbo　Basic程序。

§3.1　数据处理循环：输入／处理／输出

多数程序读入数据或输入（input），然后以某种方式处理，最后产生信息，叫做输出（output）．输入可以用键盘以一系列条目输入．例如：



输入也可以从磁盘文件或其它存贮媒介读入：



磁盘上可接收到的记载数据的帐目文件

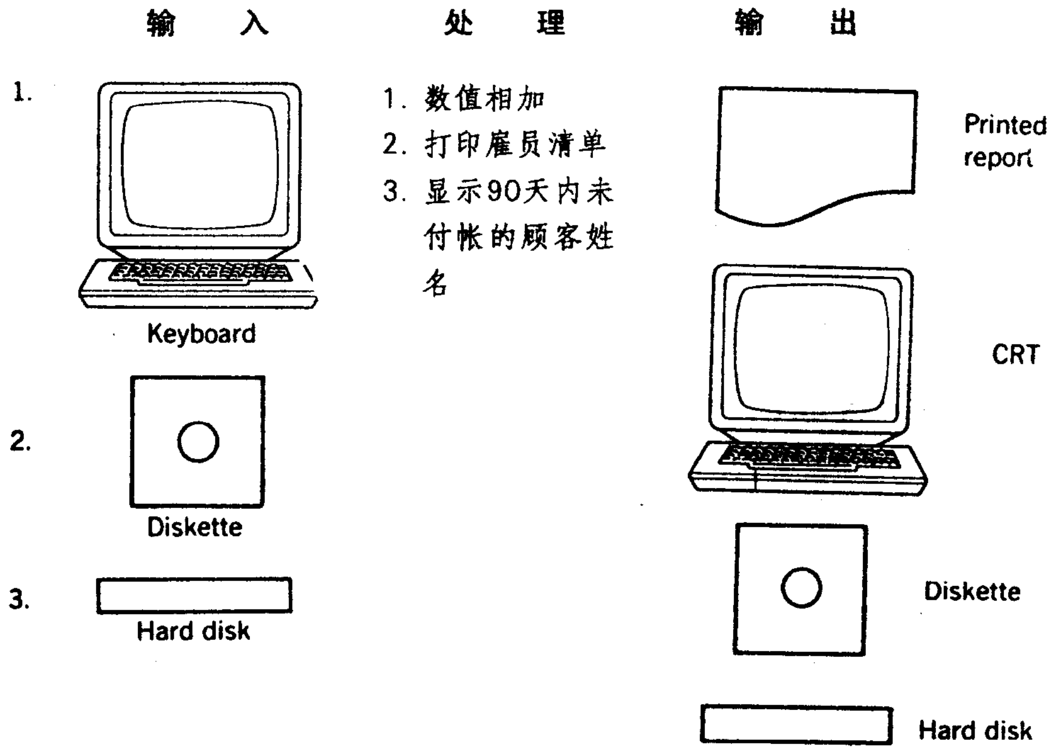
处理输入的方法有：

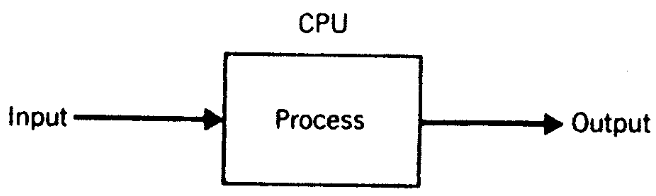
(1) 进行算术运算。

(2) 根据输入内容的不同进行不同类型的操作。

例如，在上一章的样例程序中，输入的时间变量乘以每小时酬金便得到周薪。输出可以是显示在屏幕上的、或打印在打印机上的、或存在磁盘或其它存贮介质上为以后处理的信息。下面提供了 input（输入）／process（处理）／output（输出）／周期的例子。

例:





在这一章中，将考虑一些基本的输入、处理和输出操作。阅读本章后，用户就能够完整地编写简单的 Turbo Basic 程序。

§ 3.2 结构化程序设计概念的回顾

所有程序都将用到结构化程序设计（structured programming）方法．我们用这种方法，是因为它们可使程序便于阅读、调试、维护和修改。结构化程序包括初始读（initial read），输入一系列数据；然后用 DO WHILE … LOOP 循环或类似的逻辑控制结构。这些结构将处理初始数据，产生输出和读附加的数据，直至所有数据已经运算完成为止。那么，所有程序的基本格式应包括:

```
INPUT    . . .            ' This is the initial read
DO WHILE 还有输入?
       { 输入处理 }
       { 生成输出 }
    INPUT    . . .
LOOP
```

这种结构的引入使程序设计避免了使用分支或转移（GOTO）指令，而它们正是使程序难以阅读和调试的原因。

记得行号在许多其它 Basic 版本中是必须的，但对 Turbo Basic 却是可选择的。总之，本书将略去行号，除非需要在讨论中指明每一行。

§ 3.3 数据结构

§ 3.3.1 计算机处理的数据类型

Turbo Basic 处理两种数据类型．一种叫变量数据，另一种为常量数据．

§ 3.3.1.1 变量

变量（variable）是可以在程序每次运行中变化的数据元素．例如，客户数，项目价格或每次程序运行或执行时作为变量输入的读人数。因此，输入数据总是变量．类似地，输出信息因依赖于输入，也是变量。

程序中变量数据元素被赋予了变量名（variable name）．例如，若有:

```
INPUT  Price
```

其中: Price 就叫做“变量名”．每个变量名对应于在计算机存贮器中数据实际存放的位置或地址。因此，如果用户输入 1.75 响应价格输入语句:

```
INPUT  Price
```

变量 Price 将含有价格: 1.75．数据元素或变量可以作为输入读入，或在程序中定义或产生为输出。

§ 3.3.1.2 常量

一个常量（Constant）是不管输入如何总保持不变的值。例如，0.08 的销售税就可以在程序中定义为 0.08 的常量。

例:

考虑一个使用变量和常量的 Turbo Basic 程序样例。假定希望读入一个薪金，并加入 10%的增加量。以下为程序摘录:

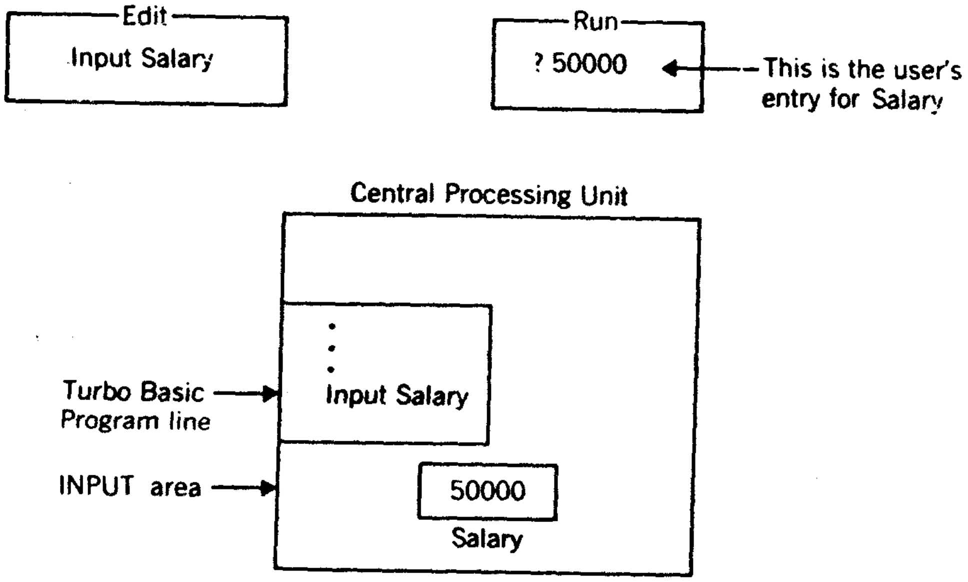
程 序 样 例

```
INPUT Salary                                ' Salary is a Variable Name
LET NewSalary = Salary + .10 * Salary       ' .10 is a constant

PRINT NewSalary                             ' NewSalary is a Variable Name
```

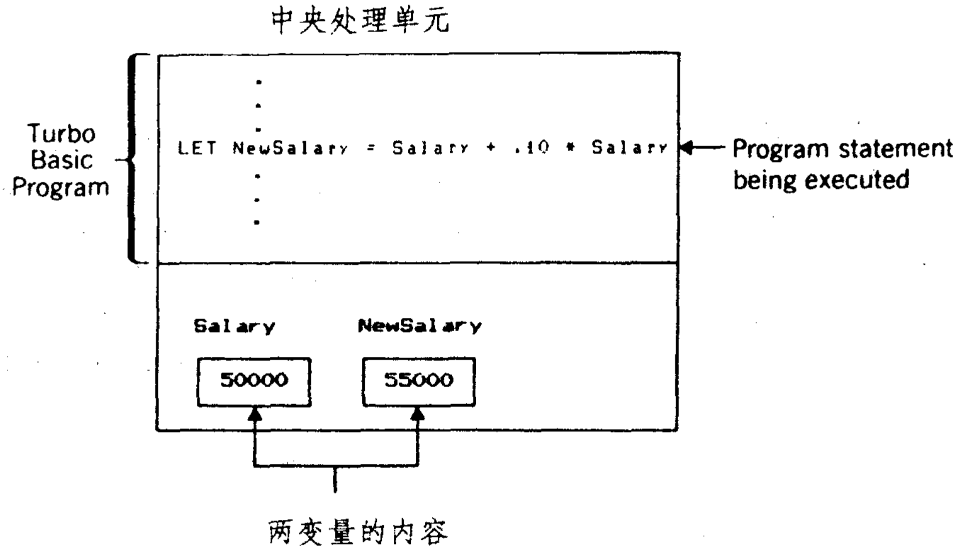
其中: 薪金（Salary）是一个输入变量；新薪金（New-Salary）是一个计算后输出或打印出的变量；数值 0.10 是常量。

程序运行时，INPUT 指令将提示用户输入数据。提示以问号(?) 出现在屏幕上的运行窗口中。执行程序运行窗口中的问号? 意味着计算机正等待用户键入数据。键入所需值或薪水数后，必须按 Enter 键（标记为↵，Enter，或在多数计算机上为 Return）将数值传入计算机。

用户输入的数值将（在 Enter 键被按下后）存放在叫 Salary 的变量中，如下所示 (RUN 窗口中的数值由用户键入):

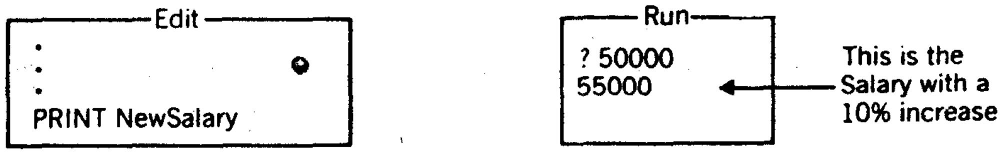


然后计算机运行程序中随后一系列指令:



Salary 和 NewSalary 是存贮被输入或由 CPU 计算得到的值的变量或符号名。

例子中的最后一条指令，PRINT NewSalary，将在屏幕上显示存贮在变量 NewSalary 中的值。



一个完整的计算所有薪金 10% 增量的程序包括：初始 INPUT 指令和一个 DO WHILE … LOOP；先处理第一个输入值，然后处理其余的输入值，直到不再有数据为止。用 9999999 的薪金值表示作业结束的条件，因为我们假设这个数或更大的数将永远是一个无效的薪金数。

```
INPUT  Salary
DO WHILE  Salary  <. 9999999
    LET  NewSalary  =  Salary  +  .10 * Salary
    PRINT  NewSalary
    INPUT  NewSalary
LOOP
END
```

记住，“<”表示“小于”。在上面，0.10 和 9999999 是常量。Salarty 和 NewSalary，如指出的，是变量。正如在第一章中指出的，用大写字母写 Turbo Basic 的保留字以区别变量和常量。

§ 3.3.2 定义变量名

§ 3.3.2.1 数字变量

每个数据元素或变量以变量名定义．定义这些名字的原则依赖于被存贮在这些变量中的数据是数字的还是字符数据的．一个数字变量（numeric varzable）含有数字，一个可选择的小数点和可选择的正、负号．这些变量可以用于算术运算．一个字符数据(alphanumeric) 或字符串变量（string variable）能包含字母或符号以及不能用于算术运算的数字．例如，一个包含地址或名字的变量被认为是字符数据．

下面开始考虑数字变量的取名原则，这些变量名表示存贮器中存贮数值的位置．

数字变量取名原则

1. 名字必须以字母开始．
2. 可以含有任意个字母和数字．
3. 可以由大写或小写字母组成，可以由这二者一起形成．
4. 点（·）可用作字分隔符．
5. 变量名中不允许有空格（如，NewSalary 是正确的，但 New Salary 错误)．
6. Turbo Basic的保留字如INPUT或PRINT不允许用作变量名(参见附录中的 Turbo Basic 保留字表)

合法数字变量名举例

```
Amount1
AMOUNT2
AmtOfSale
Amt.Of.Sale
```

建议下面的内容作为数字变量取名的编程指南．如下所述的编程指南将贯穿全文．它们的使用将使程序更易于阅读、调试和修改．

变量名的编程指南

1. 用有意义的名字定义变量．例如，Amount1作为一个数量变量就比用A1好．
2. 为了区别指令与变量，规定指令中的所有字母都大写，而变量仅有第一个字母大写．若变量名含有两个字，如 NewSalary，则大写每个字的第一个字母．因此，变量将使用这些的组合．这种规定使得容易在程序中查找变量．

例:

```
INPUT  Salary
NEWSalary  =  Salary  + 0.10 * Salary
PRINT  NewSalary
```

3. 若使用含有几个字的变量名，如上例中，建议每个字用一个大写字母或用点（·）分隔每个字。有许多方法给变量取名：

例:

```
Social.security.number
Social.Security.Number
SocialSecurityNumber
Amt.of.Sale
AmtOfSale
Amt.Of.Sale
```

建议采用上面取名规定中的一种，并且经常使用它。我们使用这儿说明的最后一种形式。

4. 使用必须多的字符个数以使变量名有意义，但记住每增加一个字符，就增加了打印错误，以及作为数据条目和键入错误的机会.

上面的编程指南不会影响程序的编辑或执行，只能增强程序的可读性和一致性。

注意: Turbo Basic是一个对字母大、小写不敏感的语言。这意味着可以交替地使用大、小写字母。因此，编译器认为大、小写字母是相同的字符。故，对编译器来说，变量 Amt 与 AMT 及 AMt 是相同的。

下面是非法变量名的例子。

非法数字变量举例

非法变量名	原　因
Unit.Price.In.$	数字变量名中不允许出现 改成 Unit.Price.In.Dolloars 即可
INDAMT	数字变量名必须以字母作开始 改成 Amt2 即可
Employee　Wages	数字变量名中不允许出现空格 改成 Employee.Wages 或 EmployeeWages 即可
Output	它是 Turbo Basic 的保留字 改成 OutputAmt 或 Output.Amt 即可(Turbo Basic 保留字详见附录 A)。

§ 3.3.2.2 字符串变量

以上的变量是用于存贮数字数据的。这些数字变量用在算术运算中。然而，还有其它变量需要用来存贮字符数据（字母、空格、和／或特殊符号，以及数字）。这些变量就叫"字符串变量"。

字符串变量的取名原则

1. 字符串变量名除了以字符 $ 结尾外，其余与数字变量取名原则相同。因而，EmployeeName $，Address $ 和 MaritalStatus $ 作为字符串变量是合法的名字。
2. 注意 Name $ 是错误的，因它是保留字。有几个以 $ 结尾的 Turbo Basic 保留字，都不能用作变量名（参见附录）。

§ 3.3.3 定义常量

正如已看到的，常量不管输入如何总是有相同的值。它在程序中直接被译为固定值。在前面例子中用于薪金增长的值.10 就是常量的一个例子。

正如 Turbo Basic 程序中有两种类型变量（数字和字符串）一样，常量也同样有两种：

常量类型	合法字符	举 例
数字常量	(a) 数字。 (b) 任选的小数点。 (c) 数的左边任选的正、负号。	46 12.74 -12.3
字符串常 量	在串常量中，用双引号(") 引起来的任何字符集都是允许的。这包括字母、数字、空格和如%，#等特殊字符。	"This is a Constant" "Input Error #412"

自我测验(一)

1. 计算机处理的两种数据类型是________数据和________数据。
2. 假定程序含有以下的程序段:

```
DO WHILE  Amt < 9999999
    :
    :
LOOP
```

变量 Amt 被键入 9999 叫________值，将产生________。

3. 下面哪些是正确的数字变量名?

   a. Age
   b. INPUT
   c. Phone.Number
   d. NameOfCompany
   e. Test Score
   f. 9th.Inning.Score

4. 下面哪些是正确的字符串变量名?

   a. $ Address
   b. Address $
   b. Name.In

5. 下面哪些是合法的数字常量?

   a. 123.45+
   b. +123.0
   c. 127
   d. .15
   e. 1,000
   f. $ 23.00

6. 下面哪些是合法的字符串常量?

   a. " This's All Folks "

b. " Error Message 123

7. (T或F) 因为TurboBasic大小写不敏感，故Amount1和AMOUNT1被认为是同一变量.
8. 执行输入指令时，运行窗口通过显示一个________，以提示用户输入数值.

9～10 题考虑下面的程序段:

```
INPUT NameIn$
INPUT Amt
LET AmtOut = Amt + 10
PRINT NameIn$, AmtOut
```

9. AmtOut是一个 和NameIn$是一个 . 在执行时，这些指令完成________操作.
10. 指令LET AmtOut=Amt+10中的10叫________.

§3.4 INPUT 语句

§3.4.1 键入数字变量

据我们所知，INPUT 语句可用来键入待处理的数据。可以把数字变量和字符串变量作为输入数据键入。

例:

```
INPUT Amount        ' Amount is a numeric variable
INPUT NameIn$       ' NameIn$ is a string variable
```

(注意: Name$是保留字，不能用作字符串变量.)

执行 INPUT 语句时，计算机提示用户输入数据. 输入提示是以问号(?) 显示在运行窗口的. 键入输入后，必须按 Enter 键才能将输入送入计算机中，以便于执行程序中的下一条语句.

可作为数字输入的字符与数字常量是相同的:

(a) 数字;
(b) 可选择的小数点;
(c) 数的左端一个可选择的+或-号.

因而，在回答"?"提示时，下述数字输入都是合法的:

115 .25 +1278 12.5 -16.3

亦即，上面的每一个数能正确地输入到象 Amount 样的数字变量中。

其它的字符，如逗号或双引号，在数字变量中都是禁止的。再次提醒，作为数字输入合法的字符在作为数字常量也合法。这意指程序本身可以包含如 115，0.25 等之类的数字变量。

考虑语句:

```
INPUT  Amount
```

程序运行时，假定提示输入变量 Amount，用户用下述任一值作响应。由于下表给出的原因，这里表示的结果是不对的。

输入数值错误举例

错误的数据	结果	解释
输入000	Amount=1	数字变量或数字常量中不允许有逗号。计算机将逗号理解成数值的结
?a	Amount=0	束不是数字范围内的有效字符。任何非数字输入在数字变量中均按 0 处理
?$12	Amount=0	$不是有效的数字字符
?12–	Amount=12	因为“–”号不在数字左边，故忽略。

如上所述，如果输入值含有一个字母或任何特殊字符（除逗号外），则数字变量将存放 0 值。记住，在这种情况下，将不会产生任何错误信息。因而，程序员必须保证用户知道允许输入哪种类型。同样，定期运行程序时，在处理输入数据前，应确认它的格式是正确的(详见第八章数据有效性处理的讨论)。

注意，键入输入时，如果没有按 Enter 键，那么可以用退格键（BackSpace）删除错误的击键。因此，在按 Enter 键前最好是仔细检查屏幕上的输入。

§ 3.4.2 键入字符串变量

字符串或字母变量也可以作为输入被送入。字符串变量名是以“$”结尾的。当输入数据送到一个字符串变量时，任何字符都是可以接收的。但注意存贮的字符串变量不能用在算术运算中，既使键入的值全是数字。因此，下面的是非法指令:

```
INPUT  Amount$
LET  AmountOut  =  Amount$  +  3    ' Amount$不可用于算术运算
```

在编译上面的指令时，将得到一个语法错误。这中因为字符串变量 Amount$ 不能在算术运算中使用。在编译期间，语法错误将如下显示在编辑窗口的顶部:

```
Error 419: String expression requires string operands
```

操作数（operand）一词在 Turbo Basic 的错误信息中就象上面例子一样经常被使用。操作数代表任何变量（字符串或数字变量）或常量（字符串或数字常量）。

上面的错误是由于打算加一个数字量 3 到一个字符串操作数 Amount$ 上去而引起的。3 不是字符串操作数，而是一个数字值。因此，它不能加到一个字符串操作数上。将在第八章中更详细地讨论语法错误。

再次提醒，Name$ 是保留字不能用作字符串变量。强调这个事实是因为 Name$ 将表示是一个"天然"的字符串变量名。

§3.4.3 没有输入任何数据而按 Enter 键会怎样

对"?"或提示的输入响应只有在 Enter 键被用户按下之后才能传送给一个变量。如果用户不小心按下 Enter 键，而没有输入任何实际数据、或按空格键后按 Enter 键，则输入变量将是:

(1) 对字符串变量来说——一个"空串"，一个空的字符集（相当于没有值）；

或:

(2) 对数字变量而言——一个 0。

例 1:

```
---- Edit ----      -- Run --
INPUT Amt              ?       <---- User presses Enter key
PRINT Amt              0       <---- Computer stores 0 in Amt
```

输入时，若按 Enter 键，则 Amt 等于 0。

例 2:

```
---- Edit ----         -- Run --
INPUT Message$             ?
PRINT Message$
```

输入时，若按 Enter 键，则 Message$ 等于空串。

§ 3.4.4 一行中输入多个变量

每条输入 INPUT 语句从终端读入一行输入．可以用一条语句输入多个变量．考虑下面的语句:

Edit

```
INPUT EmpName$, Amt1, Amt2
```

执行这条语句时，计算机希望在一行上输入三个数据．输入的每个数据间应用逗号分隔开．逗号后的空格是可送择的．假定按如下形式回答提示:

Run

```
? John Wayne, 15.5, 26.5
```

按 Enter 键后，计算机将含有下面内容:

中央处理单元

EmpName$	Amt1	Amt2
John Wayne	15.5	26.5

用一条 INPUT 语句可以读入任何多的变量，只要它们用逗号分隔开．

读入多个变量的输入语句要求所有输入在 Enter 键被按之前键入．看看下面的例子

:

```
INPUT  EmpName$, Amt1, Amt2
DO WHILE Amt1 < 9999        'Amt1 为 9999 时，终止循还
      :
      :
      INPUT  EmpName$, Amt1, Amt2
LOOP
```

可以测试三个输入变量中的任何一个作为结束值，但选择 Amt1 中的一个值 9999 作为结束值以标志条件结束。键入每一个输入行时，既使是结尾行，也必须包含所有三个变量。

```
John Wayne, 15.5, 26.5
    :
```

```
        ⋮
No Name, 9999, 9999        ← 结尾行
```

假定按下面替代方法键入输入:

```
—— Edit ——
INPUT EmpName$
INPUT Amt1
INPUT Amt2
```

这里的三条语句要求三条不同的输入行键入 EmpName$, Amt1 和 Amt2. 运行时, 上面的语句将产生三次提示, 每次在不同的行上. 用户必须用下列形式回答每一个提示:

```
—— Run ——
? John Wayne
? 15.5
? 26.5
```

§3.4.5 用问号? 对输入到字符串变量的值定界

考虑下面的例子:

```
INPUT CustNmae$
PRINT CustNmae$
```

在"?"提示下, 假定输入:

```
? Newman, Paul
```

输出 CustName$ 将显示为:

```
Newman
```

记得逗号是用于分隔输入变量的. 因此, 一旦计算机碰上逗号, 它就假定变量 CustName$ 被键入的数据是完整的。输入本身含有逗号的字符串数据, 就要使用问号确定值的界限。

```
INPUT CustNmae$          ? "Newman, Paul"
PRINT CustNmae$          Newman, Paul
```

同样，如果一个变量含在引导或最左边的空格，也必须以问号输入。例如，输入“　ABC”，有两个引导空格，就必须用问号来确定其值的界限。

例 1a:

INPUT　CODE1a $　　　　? ABC

CODE1a $ 按如下形式存放:

A	B	C

例 1b:

INPUT　CODE1b $　　　　? "　ABC"

CODE1b $ 按如下形式存放:

		A	B	C

然而，如果在字符串变量中贮存有结尾或最右边空格，就不要求用问号来确定值的界限了。假如通过输入 ABCb̸b̸(b̸表示空格)响应语句: INPUT　CodeX $，则 CodeX $ 贮存的输入值为 ABCb̸b̸. 因而，既使不使用问号，尾空格也是输入值的一部分。

例 2a:

INPUT　CODE2a $　　　　? XYZb̸b̸

CODE2a $ 按如下形式存放:

X	Y	Z		

例 1b:

INPUT　CODE2b $　　　　? "　XYZ　"

CODE2b $ 按如下形式存放:

		X	Y	Z		

贮存在 Code2a $ 和 Code2b $ 两变量中的值的不同将影响字符串处理中的比较、打印输出和其它方面。

简单地说，输入到字符串变量中的字符串数据可以有或没有外层问号。如果字符串不含有引导空格或不嵌有逗号，则贮存在字符串变量中的值将是与实际相同的，而不管是否使用了问号；若贮存的字符串值中含在引导空格，或结尾空格，或嵌有逗号，它就必须用问号包围。结尾空格将被存贮，既使不使用问号。

自我测验(二)

考虑下面为问题1—6的程序段:

```
INPUT  MilesTraveled, NumGallons
MilesPerGallon = MilesTraveled / NumGallons    ' /为除号
PRINT  MilesPerGallon
```

1. MilesTraveled称为________。
2. (T或F) MilesTraveled和MILESTRAVELED在程序中被认为是同一个变量。
3. 输入语句INPUT要求在一行里输入________个数据，并用________分隔开。
4. 执行INPUT语句时，运行窗口将含有一个________，它是作为被输入的________符号。
5. 为了使键入的值传送到计算机中，________键必须被按下。
6. 如果键入“12__，ABC”作为对提示的回答，这样做的两个错误是________和________。

对7~9题，考虑下面的程序段:

```
INPUT  Length, Width1          ' Width为保留字
DO WHILE Length > 0
    LET Area = Length * Width1      ' * 为乘号
    PRINT Area
    INPUT  Length, Width1          ' Width为保留字
LOOP
END
```

7. 这是一个叫做________程序的例子程序，这类程序不含分支或GOTO语句。
8. 在上面程序中，当________时，作业结束条件满足。
9. (T或F) 上面程序中有两条输入语句，它们二者是相同的。
10. (T或F) 如果输入: 111 5th Ave., Ny, Ny 10036 到一个名为Address1 $ 的变量中去。这个值就必须用引号引起来，因为它含有逗号。

对题11~15，考虑下面语句:

```
INPUT  EmpName $
PRINT  EmpName $
```

如果下面数据是作为EmpName$的输入而键入的，打印结果会是什么？（假定定位正确）

11. ? WASHINGTON, GEORGE
12. ? "WASHINGTON, GEORGE"
13. ? (未键入数据便按回车)
14. ?   ABC
15. ? "   ABC"
16. 找出（a），（b）间的区别：

```
(a) INPUT  Amt1, Amt2
(b) INPUT  Amt1
    INPUT  Amt2
```

下面的指令（17～20题）会产生语法错误吗？并解释原因。

17. Amt$ = "123"
    Total = 500 + Amt$
18. Code1$ = 120
19. HoldIt = "123"
20. Name$ = "Paul"

§3.5 赋值语句

§3.5.1 使用赋值语句执行基本算术运算

§3.5.1.1 格式

在 Turbo Basic 中，算术运算是由 LET 语句执行的，LET 语句也叫赋值语句。这条语句的形式是让一个变量等于一个如下的计算值：

```
[LET]  变量  =  表达式
```

上面框中的语句被称作"语法格式（Format）"。语句的语法格式为它的使用指定了正确的规则。如，字 LET 用方括弧括起来，表示它是可选择项。格式中的表达式由能计算出结果的运算符，操作数（变量和常量）组成。

表达式能用如下的基本运算符执行关于数字变量和数字常量的运算。

赋值语句中的基本运算符	
+	加法
−	减法
*	乘法
/	除法
^	指数

所有变量和常量，包括“=”左边的结果变量都必须是数字的。用作指数运算的符号“^”，是主键盘中数字 6 上的一个特殊字符。

例

合法的赋值语句	解释
1. LET　Total=Amt1+Amt2	等号左边的变量 Total 将被赋予 Amt1 和 Amt2 之和。
2. LET　Wages=Hours * Rate+overtime	等号左端的表达式含有多个算术运算符。这里，Wages 将等于 Hours * Rate+Overtime 的计算结果。
3. LET　Tax=0．05 * Price	赋值可以用来使一个变量等于一个含有变量和常量的表达式。这里，变量 Price 乘以数字常量 0．05，结果放在变量 Tax 中。
4. LET　Area=3．14 * Radius^2	这条语句是按公式 Pi.R^2 (Pi=3.14 和 R 为半径) 计算园面积，并把结果放在数字变量 Area 中。
5. Commission=Sales * .10	赋值语句中的字“LET”是可以选择的。

下面为非法的赋值语句及其错误原因：

非法的赋值语句	错误原因
1. LET　.0751 * Salary=SocSecTax	在 LET 后，即等号左端必须是一个变量，而不能为一个算术表达式。正确的语句为：LET　SocSecTax=.751 * Salary

2. LET NewSalary= $ 1, 000+Salary	$ 1, 000 含有一个 $ 符号和一个逗号，不是合法的数字常量。正确的语句是：LET NewSalary=1000+Salary
3. LET Overtime=2Hours	如果 2Hours 是指 2 倍 Hours，则乘号"*"不能省（LET Overtime=2*Hours)。另外，由于所有变量都必须以字母开头，故 2Hours 不是一个数字变量。
4. LET Bomus $ =Amt/10	由于 Bonus $ 是一个字符串或字母变量，故它不能是算术运算的结果。

在算术运算符两侧空格是可选的。但，它们使语句容易阅读。语句 LET A = B * C 就是优于 LET A=B*C，尽管二者都正确。

§3.5.1.2 完成算术运算的程序

问题 1

考虑一个读入一系列摄氏温度值，并打印出相应华氏温度值的程序。输入一个摄氏温度值为 999（即作为结束值）时，结束程序。回忆一下，结束值表示不再有数据需要处理了。结束值是在所有"真正"数据被输入并处理完后，才输入的。这个值不应是需处理的"真正"值本身。计算华氏温度的公式为：

Fahrenheit=9/5Celsius+32.

注意，所有的程序都是结构化的。这表明应用一条初始输入语句INPUT读入第一个摄氏温度值；然后用 DO WHILE … LOOP 循环处理该值并读入其余值，直至没有输入为止。这个问题的伪代码和程序如下。本例中将用基本打印语句 PRINT 在屏幕上输出结果。关于打印的其它可选项将在本章的下一节讨论。

伪代码

```
START
    读入一个摄氏温度值
    DO WHILE 还有输入?
        计算华氏温度值
        打印结果
```

```
        读入一个摄氏温度值
    END  DO
    打印结束信息
STOP
```

标准的伪代码应是以 START 开始，STOP 结束的。在 DO … END DO 块中的指令是逻辑控制序列的一部分。用在每条指令中的字不需要遵循任何程序设计语言的语法，因为伪代码作为一种规划工具是独立的语言。

Turbo Basic 程序

```
' Line number are optional with Turbo Basic
10  INPUT Celsius
20  DO WHILE Celsius < 999
30       LET Fahrenheit = 9/5*Celsius + 32
40       PRINT Fahrenheit
50       INPUT Celsius
60  LOOP
70  PRINT "That's all filks"
80  END
```

在上面的程序中，999，9，5 和 32 是数字常量，Celsius 和 Fahrenheit 是数字变量。"That's all folks!"为字符串常量。因分数 9/5 可以写成 1.8，于是行号为 30 的语句还可写成：

```
LET Fahrenheit=1.8*Celsius+32
```

在程序被编译完且无语法错误后，就准备运行它了。计算机执行输入语句 INPUT 时，一个提示符？显示在屏幕的运行窗口上。它意味着计算机正等着输入。如果在光标处输入 25：

```
┌─ Run ─┐
│ ? 25  │
└───────┘
```

然后必须按 Enter 键，使输入的值送入计算机内。在计算机内计算 Fahrenheit 等价于摄氏 25 度。每一行上希望的输入必须是不带任何如逗号或美元符号之类的编辑符号的

数字形式。

打印指令 PRINT 将打印 77 (9／5×25+32)。下一条行号为 50 的 INPUT 指令将使提示符再次显示，以要求输入另外的摄氏温度值。这一系列指令或循环反复地继续，直至值为 999 的摄氏温度被送入为止。结束值 999 结束循环，并使结束常量或信息“Thats all folks! ”打印。图 3.1 是该程序的全屏幕显示。

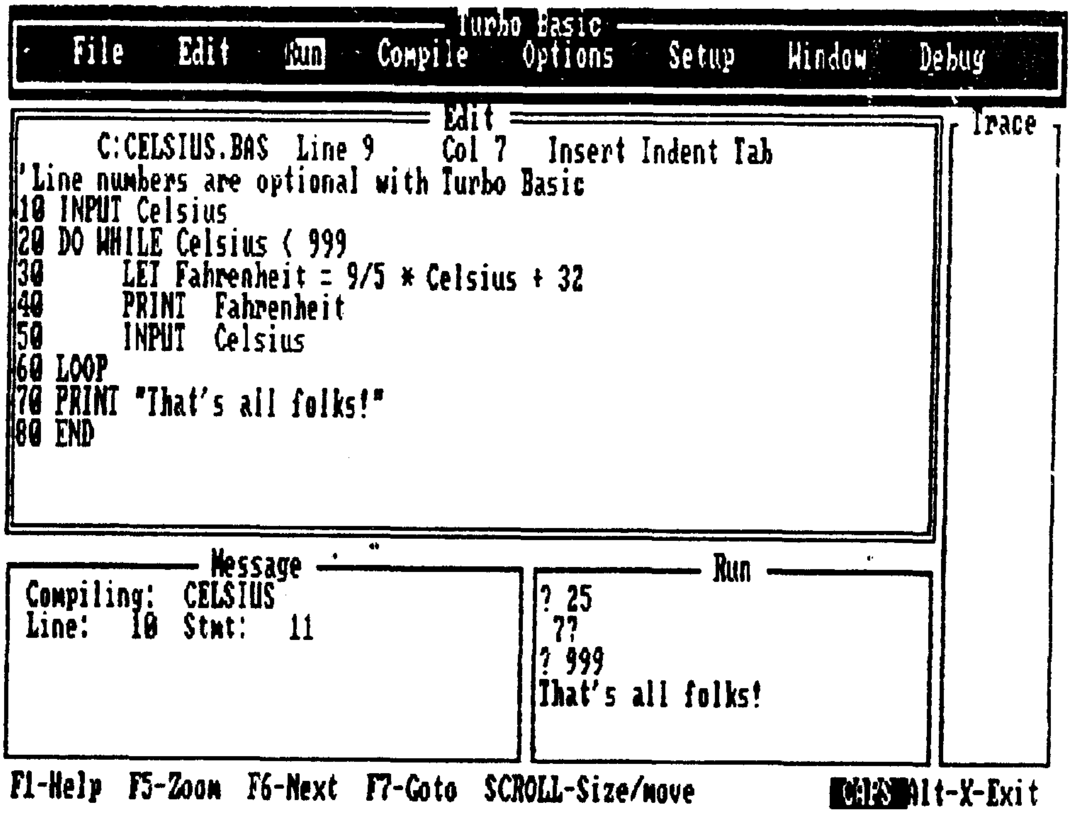


图 3.1 摄氏温度到华氏温度转换程序的屏幕显示

如果用户自己编写和运行这个程序，窗口显示可能与图 3.1 中相似。但有些项可能不同，这是因为它们取决于进入程序的方式。例如，若不在 Caps (大写)模式下输入程序，则字 CAPS 就不会显示在帮助行上。注意，屏幕上的这些细小区别并不影响程序执行。

在上面的例子程序中应注意:

1. 使用了可选择的行号。

   上面的例子程序使用了行号。这些行号使我们在叙述时便于指明指令，并且使得调试期间便于单步执行程序。然而，它们却不是必需的。在初始程序中可能想使用它们，然后确定在调试时证明它们是否有用。第八章调试程序部分将更深入的讨论行号的作用。

2. 将DO … LOOP循环体中的指令缩进以便于阅读和调试。

   语句的缩进便于理解程序的结构。缩进通过按Tab键很容易实现；Tab键为:

它位于键盘的左边。或者，用户可以每次简单地按固定数量的空格数。

问题 2

读入各顾客的应付款，并打印含有 5%销售税的结算额（balance due）。输入一个为 0 的数值时，停止运行。0 将是一个结束值。伪代码和 Turbo Basic 程序如下。

伪代码

```
START
    读入一个应付款
    DO WHILE 还有输入?
        计算结算额
        打印结果
        读入应付款
    END DO
STOP
```

Turbo Basic 程序

```
' This program Calculates a Sales Tax and adds it to a an amount
INPUT Amount
DO WHILE Amount <> 0
    Balance = Amount + .05 * Amount
    PRINT Amount
    INPUT Amount
LOOP
END
```

这个程序中，一个 Amount 在初始输入语句 INPUT 中输入。DO WHILE …LOOP 循环通过加上 5%的税计算结欠金额，然后打印它，最后读入其余的数，直至作为结束或作业结束值的 0 输入为止。Amount< >0 意指："Amount 不等于 0"，即小于或大于 0。

注意，这个程序里没有使用行号或赋值语句中的可选择字 LET。还有，本程序以一

个简短地描述其功能的注释行开始。

赋值语句还可以写成如下：

```
Balance = 1.05 * Amount
```

该例子的全屏幕程序和运行情况如图 3.2 所示。注意，在 Turbo Basic 程序中，结束语句 END 是可选的。

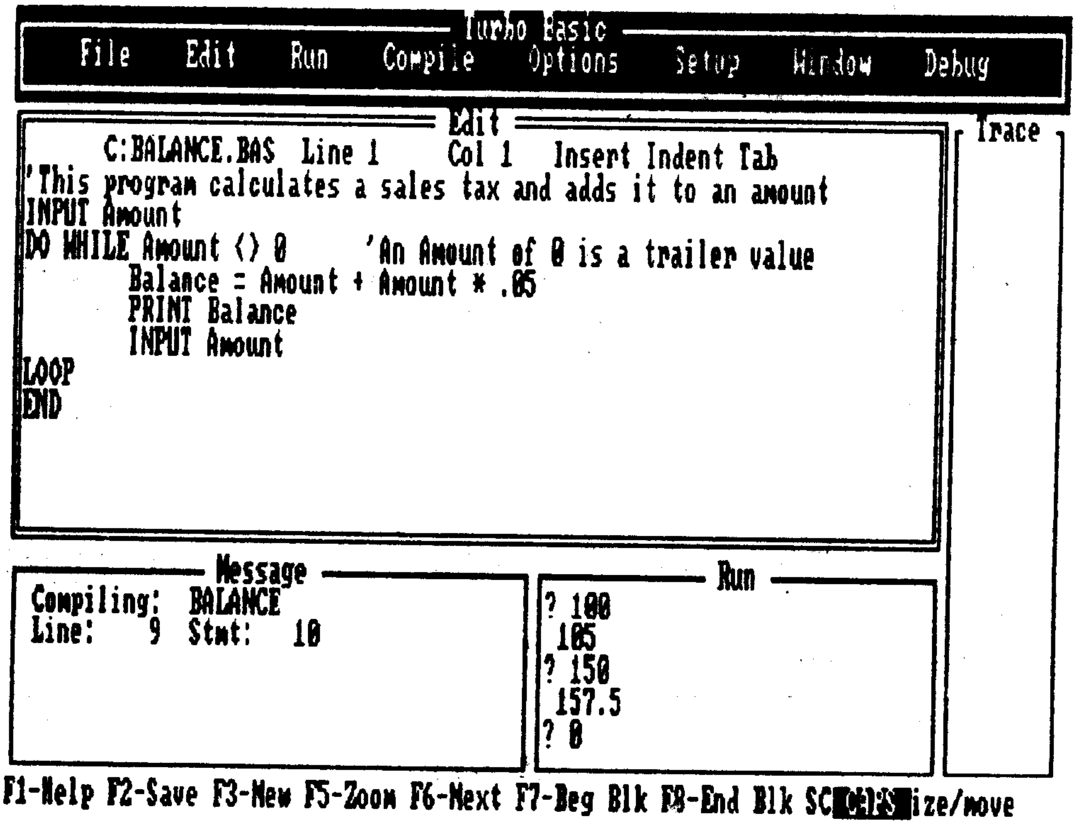


图 3.2 计算销售税并将其加到应付款的程序

自我测验(三)

1～3题的答案要求有完整的Turbo Basic程序。每个题中，程序要能够处理大量的输入变量。所有程序都应是结构化的，并事先用伪代码规划程序。

1. 编写一个程序读入一次旅行所用的汽油加仑数和行驶的英里数。打印每加仑行驶多少英里。
2. 编写一个程序读入销售额，然后按销售额的15%打印佣金。
3. 编写一程序，读入星期一到星期五的日销售额，并打印出一周的销售总和。

§ 3.5.1.3 算术运算的优先级

算术运算中操作的顺序可能影响计算结果．看下面的程序:

```
INPUT Amt1, Amt2, Amt3
DO WHILE Amt2 <> 99
    LET Total = Amt1 - Amt2 * Amt3
    PRINT Total
    INPUT Amt1, Amt2, Amt3
LOOP
END
```

假定输入给变量 Amt1，Amt2 和 Amt3 的初始值分别为 10，5 和 2．我们需要确定计算机是下述哪种方式完成 Amt1－Amt2 * Amt3 ，即等于 10−5×2 的运算:

(a) (10−5) ×2＝10

或:

(b) 10− (5×2) ＝0

在 (a) 中，计算机先作减法运算，后作乘法，结果将不同于 (b) 中先作乘法所得到的结果．

在多数程序设计语言中，计算机使用的运算规则如下:

> **运算规则**
>
> 1. 算术运算遵循如下次序:
>   a. 首先指数运算．
>   b. 其次乘法和除法运算．
>   c. 最后加法和减法运算．
> 2. 如果一条指令中含有二个或多个同级运算，则它们从左至右依次执行．
> 3. 括号的使用优于所有其它层次规则．

再回头看一下前面的赋值语句:

```
LET Total= Amt1 - Amt2 * Amt3
```

根据上面的规则，首先完成乘法运算，然后进行减法运算．也就是，Total 在算术运算执行时将有下面的值:

10 - 5 * = 10 - 10 = 0

如上如述，括号的使用优先于基本层次规则．因而，为了得到 Amt1−Amt2 乘以 Amt3，就应该写成：

```
LET Total= (Amt1−Amt2) *Amt3
```

例：

读入某班每个学生的三门成绩，然后计算他或她的平均成绩．伪代码和Turbo Basic程序如下：

伪代码

```
START
    读入三门课的成绩
    DO WHILE 还有输入?
        计算平均值
        打印平均值
        读入三门课的成绩
    END DO
STOP
```

Turbo Basic 程序

```
INPUT Exam1, Exam2, Exam3
DO WHILE Exam1 <> 999
    LET Average = (Exam1 + Exam2 + Exam3) / 3
    PRINT Average
    INPUT Exam1, Exam2, Exam3
LOOP
END
```

LET 或赋值语句中的括号是为了得到正确的运算次序而加的．下面语句将产生错误的平均值：

非法的计算平均值程序:

```
LET Average = Exam1 + Exam2 + Exam3 / 3.
```

按照运算规则，如果漏掉了括号，那么仅有 Exam3 被 3 除。故平均值 Average 就等于: $Exam1+Exam2+\frac{Exam3}{3}$.

简言之，当一个运算或一系列运算应首先完成时，就应用括号将其括起来。对于上例，想让三个成绩相加在除法完成前进行，就必须使用括号。

§ 3.5.1.4 被 0 除将导致错误

下面的程序将产生一个语法错误:

```
INPUT Amt1
Amt2 = Amt1 / 0
PRINT Amt1
```

语法错误显示成:

```
Error 11: Division by zero"
```

且光标将指向错误所在行第二行(参见图 3.3)。

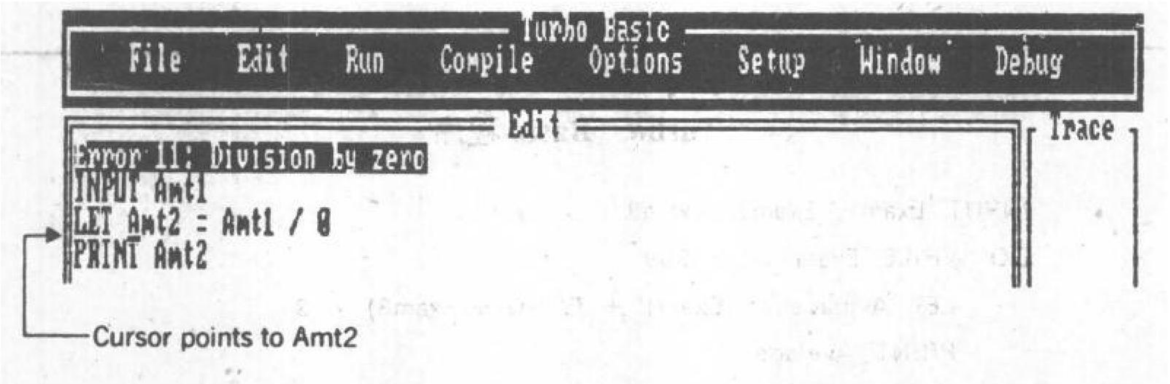


图 3.3 被 0 除导致的错误

同样，下面也将出现相同的语法错误。

```
INPUT Amt1
Amt2 = Amt1 / T
PRINT Amt1
```

由于程序中 T 没有初值，于是认为它的值为 0。正如所看到的，打算被 0 除是非法的算术运算。同理，如果 T 被输入为 0，下面也将导致一样的语法错误：

```
INPUT Amt1, T
Amt2 = Amt1 / T
PRINT Amt1
```

自我测验(四)

指出下面每条语句的结果：

```
1.  Total1 = 4 + 2 / (5 - 3)
2.  Total2 = (4 + 2) / (5 - 3)
3.  Total3 = 4 * 5 + 2
4.  Total4 = 3 + 4 * 2 - 6 ^ 2
```

指出下面每条语句的错误：

```
5.  C = 2A + D
6.  LET A + E = G
7.  Total = 0.05 * F
8.  PRINT "THE ANSWER IS , F
```

各写一条 Turbo Basic 语句完成下面的运算：

9. $A = \dfrac{3+C}{D+7}$

10. $S = \dfrac{(T-3)^2 - R}{6}$

§ 3.5.1.5 得到运行总和

正如已看到的，一条赋值语句能用来完成算术运算并将结果放在指定的变量中。例如，语句：

```
LET Net=Price-Discount
```

就是从 Price 中减去 Discount，并把结果放在 Net 中。

赋值语句作两件事：

> **赋值语句所完成的功能**
>
> 1. 完成等号右边指定的计算.
> 2. 把计算结果放入等号左边指定的变量名中.(因而, 紧接着可选字 LET 的就必须是一个变量名, 而不能是算术表达式, 即 LET B+C=A 是非法的.)

考虑下面的语句:

```
LET Counter = Counter 1 1
```

这条语句不是数学表达式, 但它是合法的 Turbo Basic 语句; 它完成:

1. 加1到Counter中;
2. 结果送回到Counter中.

因此, 若 Counter 有一个初值 10, 则将它加上 1, 并且把 11 放入 Counter 中; 即, 语句: LET Counter=Counter+1 的作用就是 Counter 加上 1. 这类赋值语句经常用来使计数器增值, 或计算总和.

一般来说, 一条这样的赋值语句是用来计算运行总和的. 一个运行总和是增加一个变量, 或加另一个变量的值. 例如, 为了累加所有读入的交易数之和, 有如下程序:

```
LET Sum = 0
INPUT Amount
DO WHILE Amount <> 999
    LET Sum = Sum + Amount
    INPUT Amount
LOOP
PRINT Amount
END
```

DO WHILE … LOOP 循环累加所有 Amount 方面的运行总和. 即, Sum 增加每次输入到 Amount 值.

注意: 程序的第一行是用来初始化变量 Sum, 使之为 0. 尽管 TurboBasic 自动设数字变量的初值为 0, 但对用作运行总和的所有变量用显示置初值为 0 是一个良好的练习. 其它版本 Basic 或别的程序设计语言可能不是自动初始化变量的, 因此建议程序设计人员养成一种习惯, 在加数字变量之前把它们置成 0. 然而, 注意象在上面的例子中变量 Amount 并不需要初始化, 因为它总是为当前输入的值.

例:

以所有薪水作为输入量读入，每一个薪水增加100后打印。伪代码和Turbo Basic程序显示如下：

伪代码

```
START
    读入销售量
    DO  WHILE  还有输入?
        将工资增加 $100
        打印新的工资
        读入销售量
    END  DO
STOP
```

Turbo Basic 程序

```
' This program increase each salary by $100
INPUT Salary
DO WHILE Salary < 9999999
    NewSalary = Salary + 100
    PRINT NewSalary
    INPUT Salary
LOOP
PRINT "End of Job"
END
```

---

§ 3.5.1.6 其它的算术运算

Turbo Basic 还可完成另外两种算术运算。它们仅对整数有效，且产生整数结果。

运算符操作	举例	
	语句	结果
/ 整除：结果问总是被截尾成一个整数（例如，4.8 的存贮结果是 4；2.5 的存贮结果是 2）	a. Y=45/3 PRINT Y b. Q=45/2 PRINT Q	 15 22
MOD 模：模运算产生一个除法运算的整形余数。Y MOD E 贮存的是 Y 被 Z 除后剩下的整数余数。	a. A = 5 MOD 3 PRINT A (5被3除时，结果为1，余数为 2；2 存于 A 中。) b. Z = 45 MOD 2 PRINT Z	 2 1

包括整数除和求模运算在内的整个算术运算优先级为：

运算符	用途
∧	指数
* 或 /	乘法、除法
＼	整数除
MOD	模运算
+或−	加法、减法

注意：模运算和整除比乘法和除法的优先级低。

例：

1. 5 MOD 2 * 3 = 5 MOD 6 = 5
2. 6 ＼ 4 / 2 = 6 ＼ 2 = 3

§3.5.2 用赋值语句给变量赋固定值

§3.5.2.1 给数字变量赋数字值

下面是合法的赋值语句，尽管它们不进行算术运算。

```
a.  LET  Sum  =  0                ' 将 Sum 的初值置为 0
```

```
b.  LET  Balance  =  Amount        ' 将 Amount 拷到 Banlance
```

可以让一个变量等于一个常量，或者用赋值把另一变量的值送给它．在例子 a 中，Sum 被置为 0；例 b 中，变量 Amount 的值放入变量 Balance 中．

还可以用赋值语句规定一变量作为常量使用：

```
LET  Pi  =  3.14
LET  Area  =  Pi  *  Radius  ^ 2
```

尽管打算让 Pi 一个固定值或常数值，但一条改变 Pi 值的赋值语句不会产生错误．另一方面，在程序中还可以定义整数常量以便其值总是固定的．任何想改变它们值的打算都将产生语法错误．下一节说明了如何定义整形常量．

§ 3.5.2.2 整数值赋给整形常量

注意，使用赋值语句整数值可以赋给整形常量．也就是，可以定义一个只能是整数值且固定不变的常量．整形常量的名字必须以一个%符号开头，以便区别于变量名．例如：

```
LET  %Salary  =  97000        ' %Salary 为其值等于 96000 的常数
LET  %Code  =  3              ' %Code 为其值等于 3 的常数
```

以%为第一个字符的定义的整形常量在程序中不能改变它们的值，而必须且只能赋给整数值．整形常量保证定义的常量不会错误地被赋予小数值，且保证在程序的其它地方不被改变．任何企图改变一个整形常量值都将导致错误．

§ 3.5.2.3 整数值赋给数值变量

以%符号结尾的数字变量被定义成整形变量．

例

```
LET  Amt1%  =  3
```

既使给定一个小数值送入到 Amt1%中，存贮在 Amt1%中的值也将是整数．

例

```
LET  Amt1%  =  3.12
PRINT  Amt1%
```

将打印出整数 3．

注意，结果是四舍五入的．因此 3.8 传入到 Amt1%中，结果将是 Amt1%＝4．

§ 3.5.3 使用带有字符串变量和字符数据的赋值语句

§ 3.5.3.1 给字符串变量赋字符数值

除了可以对数字变量赋值外，赋值语句还能用于对字符串或字符数据变量赋值。想一想，字符串变量总是以字符 $ 结尾的。在一条赋值语句中赋给字符串变量的值总是用双引号(" ") 引起来的。下面的是合法的赋值语句:

```
LET PresidentName$ = "Washington
LET Code$ = "A-OK"
PRINT CodeOut$ = CodeIn$
```

这里，LET 也是可有可无的。最后一条指令把 CodeIn $ 的值拷贝给 CodeOut $.

如上所述，赋值语句中的所有变量都必须是数字的或字符串变量。如果在赋值语句中不同数据类型混合使用，将出现下面错误:

a. *Error 418: Numeric expression requires relational operator*

```
LET X = N$
```

b. Error 419: String expression requires string operator

```
LET Y$ = X + 3
```

§ 3.5.3.2 利用赋值语句对字符串变量和字符串进行操作

算术运算符"+"可以用在赋值语句中以组合或连接字符串。

例 1.

Edit

```
INPUT LocalPhoneNo$
LET PhoneNo$ = "516-" + LocalPhoneNo$
PRINT "Phone No = "; PhoneNo$
```

Run

```
? 560-5716
PhoneNo = 516-560-5716
```

例 2.

Edit

```
INPUT LastName$
LET LastName$ = "MR/MRS/MS " + LastName$
PRINT LastName$
```

```
                                   Run
? LEE
MR/MRS/MS LEE
```

例 3

```
                                   Edit
INPUT FirstName$
INPUT LastName$
LET FullName$ = FirstName$ + " " + LastName$
PRINT FullName$
```

```
                                   Run
? JOE
? BROWN
JOE BROWN
```

注意，只有加号"+"能够用在这种字符串处理方式中。第十二章将更详细地讨论字符串处理。

赋值语句的编程指南

1. 如前所述，赋值语句中的字LET是可选的。在所有程序中，最好的是都有或都省略掉。
2. 算术运算符（=，*，+等）两端的空格不是必须的，但如果经常使用会使程序便于阅读（如，Amt_ = _5_ * _Price 就优于 Amt=5*Price）。

自我测验(五)

下面语句中，若存在错误，试指出来。

1. LET Diameter = 2Radius
2. LET Length + Width = Area
3. LET Average = Var1 + Var2 + Var3 / 3
4. Name$ = "Alfred"
5. LET %Pi = 3.14
6. Amt1 = Amt2 + -2 * Discount
7. Num1 = -Num2

给出题 8～9 的结果。

8. ```
LET Amt1 = 10
LET Amt2 = 5
LET Amt3 = 6
LET Amt4 = 2
LET Amt5 = 3
LET Result1 = (Amt1 / Amt2 + Amt3 / Amt4) * Amt5
PRINT Result1
LET Result2 = Amt1 / Amt2 + Amt3 / Amt4 * Amt5
PRINT Result2
```
9. ```
LET A = 0
LET B = 5
LET C = B/A
```
10. (T或F) 加一个数到总数上的常用方法是:

```
Total = Total + Amount
```

11. (T或F) 整形常量是不带小数点的正或负数.

计算题 12～20 的结果. 已知:

A=3	E=3
B=4	F=6
C=5	G=3
D=2	

12. `LET Z = (A + B / 2) + 6.5`
13. `LET Z = 3 * (A^b) / C`
14. `LET Z = (A / (C + 1) * 4 - 5) / 2 + (4 MOD 3 \ 3)`
15. `LET Z = D + E^2`
16. `LET Z = D^E^F`
17. `LET Z = F / E / D`
18. `LET Z = D * E + 2.5 * D + F`
19. `LET Z = C MOD G * 2`
20. `LET Z = C \ G`

对于题 21～24, 要求写出完整的程序. 在每一题中, 程序应能够处理大量的输入变量. 各程序应是结构化的, 并先用伪代码规划程序.

21. 写一个程序读入雇员的名字、毛收入和税, 然后显示每个雇员的名字和他或她的净收入, 这儿净收入等于毛收入减去税.
22. 读入星期一到星期五的日销售额, 显示日平均销售额.

23. 读入顾客名字，所购货物名称和数量，及每件货物的单价。显示每个顾客的名字，他或她所购货物的名称和相应的结欠金额。

24. 在任意给定年结束后，计算投资值的公式是：

$$V = A\left(1 + \frac{I}{100}\right)^{Y}$$

其中：V＝年末时的投资值.

A＝投资额

I＝作为一个整数的指定的投资利率

Y＝投资额投资的年数

写一个程序读入 A，I 和 Y 的值，然后计算、并显示 V.

§3.6 打印语句 PRINT 和 LPRINT

§3.6.1 一般格式

用 Turbo Basic 得到输出的两种基本方式是：

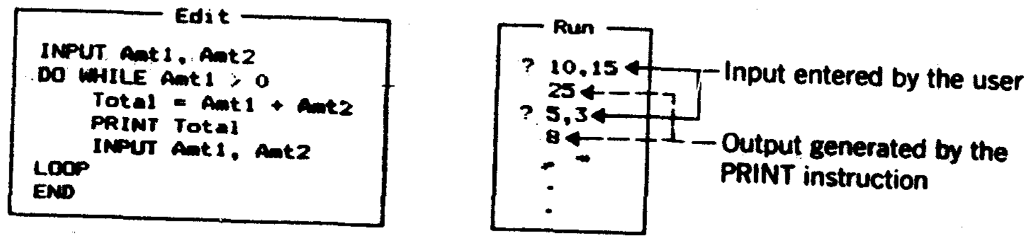
PRINT ——在屏幕上显示输出

LPRINT——在打印机上打印输出

PRINT

运行窗口将包含由 PRINT 指令产生的所有输出。这些输出将与用户键入的输入交替显示。

例 1 用 PRINT 显示结果



LPRINT

如果输出以硬拷贝或打印形式保存，就要用 LPRINT 指令代替 PRINT 指令。

例 2 用 LPRINT 打印结果

```
Edit
INPUT Amt1, Amt2
DO WHILE Amt1 > 0
   TOTAL = Amt1 + Amt2
   LPRINT Total
   INPUT Amt1, Amt2
LOOP
END
```

```
Run
? 10,15
? 5,3
```

Only input appears here

打印输出

```
25
8
```

用户将会发现后一种方法更为重要，因为在打印机上打印输入以及输出使用户有了关于每一结果含义的硬拷贝记录.

使用 PRINT 和 LPRINT 的规则是相同的，因此一起考虑.

PRINT 和 LPRINT 的格式如下:

{ *PRINT* / *LPRINT* } [表达式表] [;]

PRINT 和 LPRINT 指令有相同的格式，也就是说，PRINT 和 LPRINT 都可以用在上面的格式中.

PRINT 和 LPRINT 格式的说明

1. 大写字是Turbo Basic的关键字，在格式中使用时要求大写字. 即，它们对编译器来说是具有特殊含义的保留字. 这里，要求字 PRINT 和 LPRINT.
2. 在上面的格式中，如"expression list"（表达式表）之类的小写字是用户定义的; 它们通常由操作数组成（操作数是程序员定义的变量和常量).
3. 所有括在方括号内的项是可选择的. 因而，上面的格式中，PRINT或LPRINT可以在一行里只有其本身而没有其它项，这将打印一个空行.
4. 用于分隔不同项的标点，如上面的分号（;），是任选的.

表达式表可以是:

I. 一系列变量和／或文字

a. 变量

- 数字（如，PRINT AmtOfSales)

- 字符串或字符数据（如，PRINT EmployeeName$）

b. 文字

- 数字（如，PRINT -12.0，15）
- 字符串或字符数据（如，PRINT "Data Entry Error"）

2. 算术表达式（如，PRINT Amt1 * 2，打印Amt1乘2的结果）.

表达式中的各项可以用分号或逗号分隔开．因而，例子中我们将使用分号作为分隔符，因为分号能够保证在一个单行里每一项直接显示在前一项的右边．本章的后面将讨论打印语句 PRINT 中逗号和其它分隔号的作用．在 PRINT 语句中，逗号或分号后的空格是任选的．

基本例子

语法格式	意　义
PRINT	在屏幕上显示一个空行
LPRINT	在打印机上打印一个空行
PRINT　Amt1	显示存贮在变量 Amt1 中的值

注意，字符串变量和常量将一个接一个地打印，除非含有空格．

例 1

语法格式	意　义
PRINT "Name is"；LastName$	在同一行上显示字符串常量"Name is"和字符串变量 LastName$ 的值．字符串常量"Name is"中"is"后面要求有空格，否则名字——LastName$ 会与"is"中的 S 打印在一起或发生冲突（如，Name isJohnson）.
PRINT FirstName$；" "；LastName$	打印字符串变量 FirstName$ 和 LastName$，并用空格分隔开．设有双引号（" "）中的一个格，这两个名字就会发生冲突，或一个紧接加一个打印．（如，JohnSmith 而不是 John Smith）

如前所述，表达式表中变量和文字要求有这样的空格作为分隔符．还有，无符号的数字变量和常量在它们的值前总打印一个空格．因此，作为分隔符的空格“ ”是不必需的．但是，在负数前就没有打印一个空格，因而就必须提供合适的空格数．

例 2:

语法格式	意　义
PRINT "Salary = "; 2500	在同一行里显示字符串变量 Salary= 以及数字常量 25000．因此，将打印出 Salary=25000. 注意在语句 PRINT、"Salary = ";25000 中，等号后没有空格，将以一个可读的方式 Salary=25000 打印，因为与字符串变量不同，无符号数或正数总是由空格开始的．
PRINT "Balance is";−15000	负数不是由空格打头的．因此字符串变量中的字"is"就必须跟一个空格以确保适当的空隙．
LPRINT "Sum Of Amit1+Amti2 = "; Total	打印字符串常量并接着打印数字变量 Total 的值，如"Sum Of Amt1+Amt2 = 25"如果 Total 是一个正数，那么文字中等号之后就不需要空格，但是其中应包括数字变量有负数值的情况．

概括地说，所有数字，包括正数和负数之后都有一个空格，但仅有正数是以空格开头的．

例 3

语法格式	意　义
PRINT Amt1;"is it"	所有数字后都跟有一个空格．因此，如果 Amt1 等于 12，则将打印"12 is it "。如果 AMT1 = −12 则要打印、"−12 is it "。在每种情况下，"is"中字母"i"前不需要空格．
PRINT Amt1; Amt2	所有数字以一个空格结尾，而正数还以一个空格开始．因此，若 Amt1 是正数，则

Amt1 和 Amt2 间就有两个空格分隔（如，12.5 456）。另外，如果 Amt2 是负数，它们之间就仅有一个空格分隔，这是因为负数不是以空格开始的（如，12.5 -4560）。

编　程　指　南

1. 了解关于正、负数前后空格的规则是很麻烦的。为了避免任何可能的冲突，在所有需打印的变量和值之间总应包含一个空格。
2. 虽然算术运算可以包含在一条PRINT或LPRINT语句中，但是建议将这样的语句分成两条语句:

```
Total = Amt1 + Amt2
PRINT "The Total is "; Total
```

这种形式就比下面的便于阅读和容易调试。

```
PRINT "The Total is "; Amt1 + Amt2
```

自我测验(六)

1. ________语句把输出显示在屏幕上。
2. ________语句把输出打印在打印机上。

在题 3～7 中,每种情况下的结果是什么?

3. LPRINT
4. PRINT
5. PRINT X; Y
6. LPRINT "The total is "; Total
7. LPRINT X + Y
8. (T或F)正数打印有引导空格，而负数没有。因此，语句:

```
PRINT "The result is "; Amt1
```

在 Amt1 为负数时会发生冲突。

9. (T或F) 数字变量后总打印有一个空格．故语句:

```
PRINT  Amt1; Amt2
```

不管 Amt1 和 Amt2 是正数还是负数，二者间都有一个空格．

10. (T或F) PRINT 或LPRINT语句中的分号使打印或显示的各项之间有两个空格．

§3.6.2 改善输出的可读性

§3.6.2.1 随变量打印文字使输出的意义更明确

一般来说，将字符串表达式和计算结果一起打印可使输出的意义更加明确。

例:

```
PRINT "The Sum is "; SUM
```

§3.6.2.2 在数据输入前为用户打印说明信息

可以用 PRINT 语句向用户提供一些有关数据如何输入的信息．当说明信息作为提示需要提供给用户时，它们就应在第一条输入语句 INPUT 前．

Edit

```
PRINT "Enter Two Amount Variables Separated By a Comma"
PRINT "Enter 999, 999 When Done"
INPUT Amt1, Amt2                'Get initial amounts
    DO WHILE Amt1 < 999         'Check trailer value
        Total = Amt1 + Amt2
        PRINT "The Total Is "; Total
        INPUT Amt1, Amt2        'Get additional Amounts
    LOOP
PRINT "End Of Job"
END
```

Run

```
Enter Two Amount Variables Separated By a Comma
Enter 999, 999 When Done
? 10, 15
The Total Is 25
? 999, 999
End Of Job
```

这两条说明性 PRINT 语句在运行开始时只执行一次，但如果希望再次执行，可以把

它们放在循环体内的第二条 INPUT 语句前；这样，在每次输入提示前都将打印这些信息。

§3.6.2.3 在 INPUT 语句中给予提示代替使用 PRINT 语句

如前所述，执行一条输入指令前，通常要向用户提示输入。由于在输入前打印信息是如此普遍，因此 Turbo Basic 允许我们使用 INPUT 语句本身显示提示信息。INPUT 语句以括在双引号中的信息或指示开始，然后是分号，最后是需输入的变量名。

例:

```
INPUT "Enter Balance Due:"; BalanceDue
```

在这个例子中,提示符问号?和要输入的 Balanceduc 变量出现在同一行里:

```
Enter Balance Due:? 115.25      ←  用户提供的数
```

如果用逗号代替分号来分隔输入值和提示符，则问号？就会消失。
对比用 INPUT 语句提供的提示:

```
PRINT "Enter Balance Due:"
INPUT BalanceDue
```

结果显示如下:

```
Enter Balance Due:
? 115.25      ←  用户提供的数
```

§3.6.2.4 在打印机上打印输入和输出

与显示输出相对应，作为打印输出，包含有实际输入和结果是经常有用的，这是为了使硬拷贝打印输出包括所有有关的信息，即输入与相应的输出。

```
PRINT "Enter Two Amounts Separated By a Comma"
PRINT "Enter 999, 999 When Done"
INPUT Amt1, Amt2              'Get initial amounts
DO WHILE Amt1 <> 999
    Total = Amt1 + Amt2
    LPRINT "Amt1 = "; Amt1
    LPRINT "Amt2 = "; Amt2
    LPRINT "Sum = "; Total
    LPRINT                    'Print a blank line
    INPUT Amt1, Amt2          'Get more input
LOOP
LPRINT "End of Job"
END
```

运行时，RUN 窗口中只出现初始信息和输入数据：

```
                         Run
Enter Two Amounts Separated By a Comma
Enter 999, 999 When Done
? 10, 15
? 5, 3
? 999, 999
```

打印纸上出现输入数据及其相应的输出：

打印输出

```
Amt1 = 10
Amt2 = 15
Sum = 25

Amt1 = 5
Amt2 = 3
Sum = 8

End of Job
```

§3.6.3 打印或显示行太长时的处理方法

§3.6.3.1 避免字回绕

打印或显示的字符数取决于不同的设备。多数监视器每行显示 80 个字符，但有一些每行仅显示 40 个。在纸上一行能打印的字符数同样取决于打印机。多数打印机每行可打印 80、100 或 132 个字符，但也还有一些其它规格。

如果在一行里显示或打印的字符数超过了行长度，信息将继续或回绕到下一行。假定程序如下，打印在每行有 80 个字符的打印机上：

```
                         Turbo Basic
 File   Edit   Run   Compile   Options   Setup   Window   Debug
                            Edit                                    Trace
     C:NONAME.BAS   Line 4     Col 5   Insert Indent Tab
Pres1$ = "George Washington"
Pres2$ = "Abraham Lincoln"
Pres3$ = "Franklin D. Roosevelt"
Pres4$ = "John F. Kennedy"
LPRINT Pres1$;" ";Pres2$;" ";Pres3$;" ";Pres4$;" were presidents"
```

输出的打印行将回绕到下一行:

> **字回绕举例**
>
> George Washington Abranham Lincoln Franklin D. Roosevelt John F. Kennedy Were presidents

如果在全屏幕显示状态下打印输出，结果也是一样的.

PRINT 和 LPRINT 语句的编程指南

建议在打印或显示输出时避免行回绕，因为这些行是难以阅读的。例如，下面产生的结果就比较容易阅读:

Edit

```
Pres1$ = "George Washington"
Pres2$ = "Abraham Lincoln"
Pres3$ = "Franklin D. Roosevelt"
Pres4$ = "John F. Kennedy"
PRINT "The following were presidents:"
PRINT Pres1$
PRINT Pres2$
PRINT Pres3$
PRINT Pres4$
```

Run

```
The following were presidents:
George Washington
Abraham Lincoln
Franklin D. Roosevelt
John F. Kennedy
```

---

§ 3.6.3.2 用 Zoom 键察看 Edit 和 Run 窗口中屏幕外的数据

§ 3.6.3.2.1 Edit 和 Run 窗口的大小受到限制

如前所述,在使用 PRINT 指令时，监视器限制了显示的大小，它一般是有 25 行，每行 80 个字符，更进一步的限制是 TurboBasic 使用了窗口而不是全屏幕，以使程序和输出能够同时显示。程序显示在标记为 Edit (编辑) 的窗口中，对输入 (INPUT) 语句的回答和显示 PRINT 输出是交替地出现在标记为 Run (运行) 的窗口里。这两窗口只用了屏幕的一部分作为它们各自的显示。

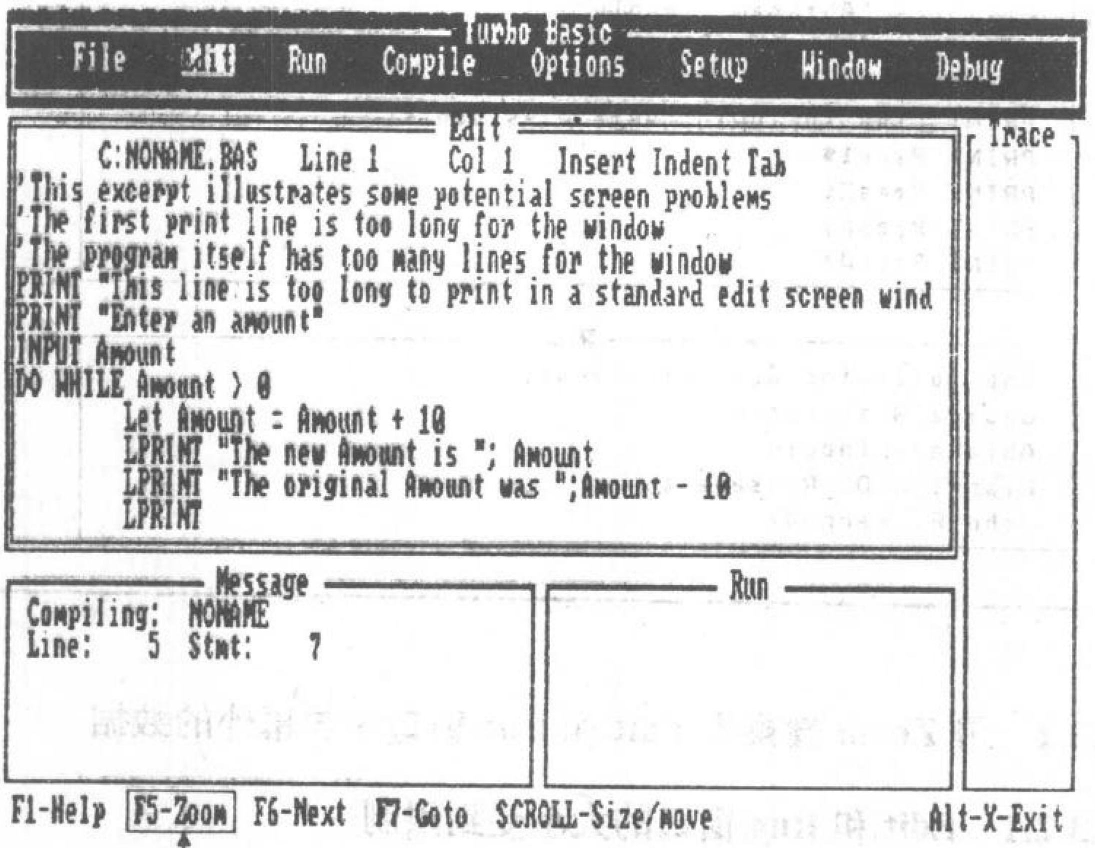
Turbo Basic 最初建立起我们所看到的那些窗口，但在第九章中将看到，用户可以把它们中的任何一个改变成所希望的任何大小。

由于编辑和运行窗口都比全屏幕小，就不可能在显示主屏幕时，随时能看到整个程序或它在运行窗口的输入／输出。现在讨论的方法可以让用户将整个屏幕用作编辑或运行窗口。

§ 3.6.3.2.2　用 Zoom 键显示全编辑屏幕或全运行屏幕

用功能键 F5(Zoom 键)放大编辑或运行屏幕以使相应的窗口使用全屏幕。也就是，在键入一程序时，用户能够放大 EDIT 窗口使程序充满整个屏幕（实际上只占有 23 行，另两行由系统使用，其中一行在顶部，另一行在最下面）。同样，在执行一个程序时，也可以放大运行窗口以便通过全屏幕交替的让输入和输出显示。

§ 3.6.3.2.3　在编辑模式下改变程序显示面积



Pressing the F5 key will enlarge the Edit window to the full screen with no other windows displayed

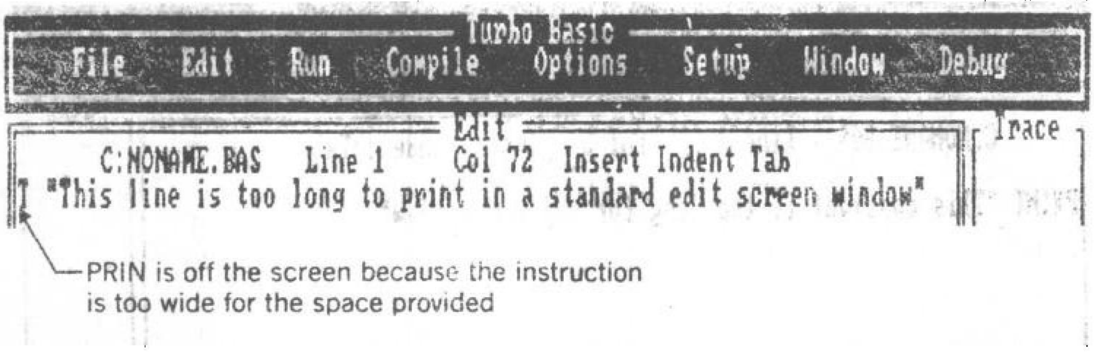
注：在当前状态下，按功能键F5可使Edit窗口放大到整个屏幕，而不显示其他窗口。

在编辑模式下，按下 F5-ZOOM 键，显示程序的编辑窗口就将占据整个屏幕．在这种方式下，可以全屏幕显示程序。要回到标准的四窗口格式，只须再按一次 F5-ZOOM 键；它是一个“乒乓开关”键。按一次 F5，若当前正在编辑模式下，则扩大 EDIT 窗口；再按一次 F5，则返回标准窗口形式。

例 假定一个程序行比编辑窗口宽许多．

考虑上述图例，第四行——打印行——与窗口相比太宽了．

在第一条 PRINT 语句中，用户可继续输入文字，但是在屏幕上不会显示出它的全部．事实上，继续键入文字时，这条指令将水平向左移动：



注：在图中箭头指向的行中，PRIN超出了屏幕显示范围。

为了看整条 PRINT 语句以及在键入程序时的整个程序，按下 F5-ZOOM 键放大编辑窗口到全屏幕的宽度和长度．若有必要，可用光标箭返回到程序的开始处．将显示下面的内容：

```
'This excerpt illustrates some potential screen problems
'The first print line is too long for the window
'This program itself has too many lines for the window
PRINT "This print line is too long to print in a standard edit screen window"
PRINT "Enter an amount"
INPUT Amount
DO WHILE Amount > 0
        LET Amount = Amount + 10
        LPRINT "The new Amount is ";Amount
        LPRINT "The original Amount was ";Amount - 10
        LPRINT
```

要回到四个分离窗口．只要再按一次功能键 F5 就行．

§ 3.6.3.2.4 放大运行窗口

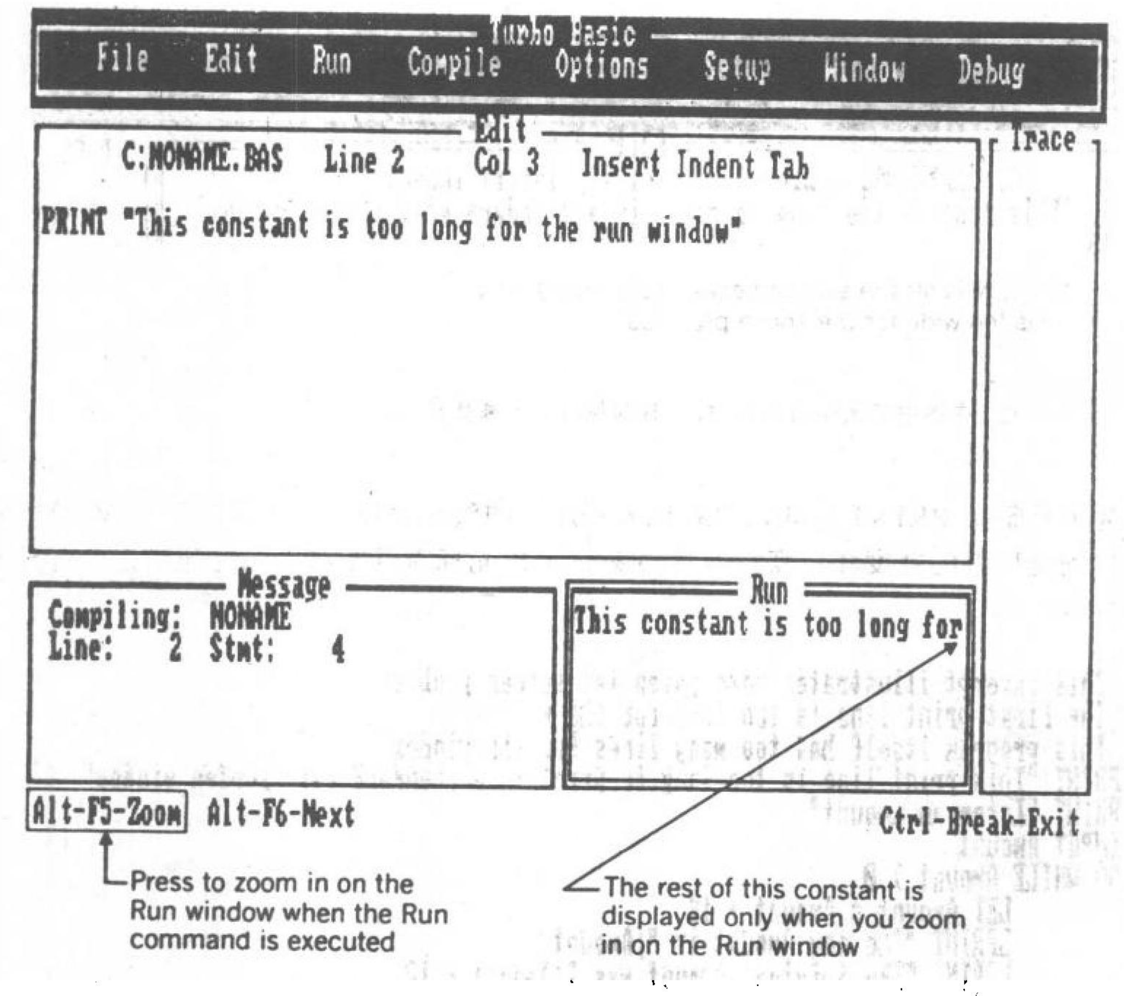
如上所述，在编辑模式下，通过按功能键 F5 可以扩大程序．同样，在运行模式下，也可以放大运行窗口．运行窗口显示输入和输出．通过按键盘左边的 Alt 键，并且在按下

的同时按功能键 F5，则运行窗口将被放大．它被命名为组合键 ALT−F5．

右下角窗口的运行窗口显示输入／输出的 29 个字符，除非它重新设置．多数时间，显示的输出行每行用了 29 个以上的字符．在运行命令模式下，用 ALT−F5 组合键放大运行窗口．使运行屏幕使用屏幕的整个长度和宽度．

例：

运行程序时，由 PRINT 语句产生的输出显示在 RUN 窗口上；此外，还显示有从键盘键入的输入．假定一个打印指令要打印 44 个字符，而运行窗口每行只能显示 29 个符，则最右边的 15 个字符不在屏幕上显示．



注：执行Run命令时，按组合键Alt−F5可使Edit窗口放大到整个屏幕．放大后，Run 窗口可显示完整的运行信息．

如上所述，在运行模式下要显示运行窗口使其充满全屏幕，就按 Alt−F5 键．在这种方式中，计算机将放大运行窗口．这意味着运行窗口将覆盖整个屏幕，使用户能够看到显示输入和输出的每行有 80 个字符，且有 23 行．Alt−F5 键也是"乒乓开关"键，因此第二次按下它俩时就关闭缩放功能，而回到标准窗口格式去．

在编辑模式下，屏幕底部的帮助（Help）行显示 F5-ZOOM，提示用户如何放大编辑窗口。同样，在 RUN 模式下，Alt-F5-ZOOM 也显示在帮助行。帮助行中的其它提示将在第九章解释。

显示长程序或数据行的编写原则

有时必须编写一些长程序行，但它们太宽以致在编辑窗口不能显示完。同样，有时也必需显示一些太宽以致不能在运行窗口显示完的输出，在这些情况下，使用 F5 缩放键检验显示。在编辑模式下，F5 缩放键将扩大程序到全屏幕；而在运行模式下，首先按 Alt 键，与此同时按 F5 键，这使从程序得到的输出显示到整个屏幕。

如果能够把长程序或显示行分成在相应的编辑和运行窗口中显示的较小长度，则希望用户这样做。若要修改编辑和运行窗口的大小，参见第九章。

---

§ 3.6.4 PRINT 和 LPRINT 语句中分隔符的功能

§ 3.6.4.1 分号作为分隔符的回顾

注意，使用分号分隔打印的字符串变量或字符串常量时，每一项直接紧挨着打印在前一项的右面。因而，除非程序员小心地加入空格，否则输出格式可能混乱。

例 1

1. Edit

```
FirstName$ = "John"
LastName$ = "Smith"
PRINT FirstName$; LastName$
```

Run

```
JohnSmith
```

2. Edit

```
FirstName$ = "Bill"
PRINT "The Name is"; FirstName$
```

Run

```
The Name isBill
```

因此，打印字符串常量和字符串变量时，各项间一定要加入空格。

例 2

```
PRINT FirstNmae$; " "; LastName$
PRINT "The Name is "; FirstNmae$
```

然而，打印数字值时，多数情况下表达式表中打印各项间不须含有空格。这是因为:

1. 所有的数字值总是带有一个尾空格的。

2. 正数（或无符号数）总是以空格开头的，负数总是由一个负号开始的.

例 3

Two spaces print here—one after 15 and one before 10

Edit

```
Amt1 = 15
Amt2 = 10
PRINT Amt1; Amt2
Amt3 = -15
PRINT Amt2; Amt3
```

Run

```
15  10

10 -15
```

One space prints here after 10 (since -15 is negative, no space precedes it)

其中: 在第一个输出行中，数据间有两个空格——一个在15之后，另一个在 10 之前.

在第二个输出行中，数据间只有一个空格——在10之后(因为-15为负数，所以无前导空格.

例 4

```
Amt1 = 20
PRINT "The Amount is"; Amt1
```

```
The Amount is 20
```

只有一个潜在的问题，就是打印一个跟有负数字符串值时. 因为负数不是以空格开始的，它们将与字符串值发生冲突，除非用一个空格作为分隔符.

例 5

```
PRINT "Amt is"; -15
```

除非在 is 中的 s 后面明确地加入一个空格，否则打印结果为: Amt is-15.

PRINT 或 LPRINT 语句中用空格分隔操作数的编程原则

尽管实事上 Turbo Basic 的正数值总是含有空格,但在表达式表的所有操作数（常量和变量）的前后最好还是包含有空格. 这能保证输出格式不发生混乱.

§ 3.6.4.2　用空格作分隔符

表达式表中用作分隔符的空格的功能与分号相同。考虑下面的举例，其中 PRINT 语句中的各项间是用空格，而不是用分号分隔。

Edit

```
Amt1 = 15
FirstName$ = "Paul"
PRINT "Amount =" Amt1
PRINT "Name Is " FirstName$
```

Run

```
Amount = 15
Name Is Paul
```

空格或分号都可以从表达式表中去掉而不影响输出；然而，去掉标点符号和空格会使程序难以阅读.

§ 3.6.4.3　逗号用作分隔符

与分号或空格相反，在 80 列的屏幕上，逗号把表达表中的每个操作数分成有 14 个字符的打印区域. 看下面的例子：

Edit

```
PRINT Amt1, Amt2
```

Run

```
15            20
```

14 positions　　14 positions

其中：各变量都将使用14个打印位，二不管其内容如何.

若一个常量或变量需要 14 个以上的输出位，它就要用多个区断. 例如，若需要输出 16 位，则要占用两个区段（或 28 个实际位置）；若要用 30 个输出位，则要占有 3 个区段(实际上为 42 个位置)，等等.

例

Edit

```
Pres1$="George Washington"
PRINT "Our First President Was "; Pres1$
```

Run

```
Our First President Was       George Washington
```

28 positions (2 zones)

逗号用作分隔符得到一个更广泛的显示区域，一般在操作数间有大量的空格．对有些应用场合来说，这可能使输出的可读性更好．

那么，在80列的屏幕上，就有五个显示区域，其中前面4个各有14个字符，最后一个区域长达24个字符．在40列的屏幕上，只有两个打印区域，第一个有14个字符，第二个有26个字符之多．

PRINT 指令中还有许多其它任选项，包括作为格式化输出的 PRINT 和 PRINT USING（同样，也有 LPRINT USING）．这些任选项将在第六章中讨论．

自我测验(七)

1. 让信息打印在打印纸上的语句是________，而让信息显示在屏幕上的语句________．
2. (T或F) 上面两条语句基本上有相同的语法．
3. (T或F) 一般来说，用逗号分隔打印元素要比用分号作为分隔符在操作数间产生更多的空格．
4. (T或F) 在第一条INPUT指令前，一般利用一条PRINT指令向用户提供如何输入数据的信息．
5. (T或F) PRINT 指令中用作分隔符的空格的实际作用就象逗号一样．
6. (T或F) 如果打印或显示的字符数超过了行长，信息就会回绕到下一行．
7. 指出面指令间的区别：

```
10 PRINT 8 +5
10 PRINT "8 +5"
```

8. (a) 中的PRINT和 (b) 中的注释有何作用？

```
(a) PRINT "Enter Name and Address"
    INPUT Name1$
    INPUT Address1$
(b) INPUT Name1$            ' Enter Name"
    INPUT Address1$         ' Enter Address"
```

仅用INPUT指令重写 (a) ,实现同样的效果．

9. 打印数字时，正数以________开头，而负数由________开始．
10. 一个打印区域包含有________个字符．

§3.7　完善程序

§3.7.1　注释

为了使程序更加清楚、可读，可以经常性地、自由地使用注释。在程序中，有两种编写注释的方法：

1. 使用注释语句REM进行注释可以（a）用一条单独的REM语句或（b）在一条指令行的末尾使用。

例 1:

```
REM   This program Calaulates Weekly Wages
Total = Amt1 + Amt2      REM 将两个输入相加
```

2. 任何行里以单引号开始一条注释。编译器将不管本行单引号以后的内容。

例 2:

```
' This program Calaulates Weekly Wages
Total = Amt1 + Amt2      ' 将两个输入相加
```

§3.7.2　每次运行前清屏

编程时，在初始注释后以 CLS（清屏）作为第一条指令是有用的，可使程序每次执行时能清除运行屏幕。

没有 CLS 指令时，运行窗口把每次运行的结果直接显示在前次运行结果的下面。这意味着，程序的各次运行或测试的结果都将一起显示在运行窗口上。为了避免这种情况，使每次运行结果显示在一个"清洁"的运行窗口上，每个程序应以 CLS 或清屏幕语句开始。

一般来说，程序以简单地描述其功能的注释开头，随后是 CLS 指令以清理运行屏幕。

§3.7.3　指令续行

我们可能要使用续行字符，一个下划线（_），以允许一条指令继续到下一行。

例:

```
PRINT "The Banlance Due For "; CustomerName$; _
         " is "; BalDue
```

如果象上面那样使用下划线，那么单个的输入可以接到续行符后面。然而，不能把一个特定的输入本身，如一个字符串常量分开。

非法情况：

```
PRINT "Try it_
          "this way"
```

§3.7.4 有关数字变量存贮十进制数的警告

程序中，数字输入有整数值、或诸如0.50，.25，.75之类的小数部分时，输出按通常的方法打印或显示。

然而，用户可能会发现，有时带有小数成分的其它类型数字在显示或打印时不同于它们的输入形式。考虑下面的例子：

```
Edit
INPUT Amt1
PRINT Amt1
```

```
Run
? 1.2
1.200000047683716
```

这种奇怪的现象涉及到计算机处理小数的方式。在计算机中，所有数字——整数以及小数——都用二进制码表示。计算机按：

$$\frac{a}{2}+\frac{b}{4}+\frac{c}{8}+\frac{d}{16}+\frac{e}{32}+\cdots$$

形式表示小数，其中由计算机计算a，b，c等等。一些数字可以用这些小数部分的项来精确地表示，如：

$$\frac{1}{2}=\frac{1}{2}+\frac{0}{4}+\frac{0}{8}+\cdots$$

$$\frac{3}{4}=\frac{1}{2}+\frac{1}{4}+\frac{0}{8}+\cdots$$

但是其它小数，如0.2或.17只能由：

$$\frac{a}{2}+\frac{b}{4}+\frac{c}{8}+\frac{d}{16}+\frac{e}{32}+\cdots$$

近似地表示。因此，数据输入后，在计算机内部以近似方式存贮。这些数字打印出来时，就看起来有点奇怪了。

3.17 打印成：3. 170000076293945

1.2 打印成：1. 200000047683716

解决这个问题有许多方法。第六章将仔细地讨论。现在，建议用户把输入限制成.5，.25，.75等等，以免发生近似问题

§ 3.8 本章辅导

§ 3.8.1 本章小结

一、 数据结构

1. 程序执行时改变值的数据称为变量数据。这包括被输入的数据、总和和计数器。
2. 在整个程序中都保持相同值的数据叫着常量数据.
3. 数据可能是数字或字符串。数字变量名由字母和数字组成，且以字母开头。字符串变量名除必须以 $ 字符结尾外其余与数字变量名相同。数字变量可以含有数字，小数点和引导符。字符串变量可以包含任何字符。表示字符的计算机编码叫 ASCII 码，是美国标准信息交换码的缩写。

二、Turbo Basic 语句

1. INPUT语句用于从键盘输入数据。
2. 赋值语句把一个值赋给一个常量或一个初即值赋给变量。赋值语句中的字 LET 是可选的。 LET T = 14.3 和 T = 14.3 是相同的。
3. 赋值语句还可以用来完成算术运算（如: T = x * y）。
4. LPRINT 语句把输出送到打印机上。
5. PRINT语句让输出显示在屏幕上.
6. 注释行可通过使用REM语句指明，或者是以一个单引号（'）开始的任何行。注释还可以用一个单引号附加到任何语句上。如:

```
T = 15.3          15.3 为 T 的值
```

三、 进行算术运算

1. 数字变量指定在赋值语句等号的左边（如，LET Total= ··· 或Total= ···）.
2. 所有算术运算都位于等号的右边。
3. 算术运算的运算次序如下:

^	指数
* 或 /	乘法或除法
\	整数除
MOD	取整数除的余数（模运算）
+ 或 -	加法或减法

所有同级的运算依次从左到右执行.

注意，括号（　）优先于一般的运算规则.

§3.8.2　本章自测

指出下面每种情况的答案:

1. AverageSales = DayTimeSales + EveningSales / 2
2. UnitPrice = Amt1 + Amt2 / Qty
3. (T或F) 格式语句中的方括号[　]意味着其中的内容在指令里是可选择的.
4. (T或F) 格式中的大写字在指令里是必须的.
5. (T或F) 运行窗口显示用户键入的输入和PRINT指令产生的输出.
6. 在运行前用于清屏的指令是________.
7. (T或F) 注释语句必须出现在注释语句行上.
8. 用于在PRINT语句的各项间产生各有14个字符的区域.
9. 变量名末的________字符表示变量含有数字整形值.
10. 变量名末的________字符表示变量含有字符（字符串）数据.

对题 11～15 写出 LET 语句.

11. 把变量Overtime Hours 和Hours的值加起来，和代替Hours.
12. 确定x英寸中的英尺数,并将商放在Feet 变量中，余数放在Inches变量中.
13. 把变量FriSales，SatSales 和SunSales 的值相加，和放到Weekend Sales中.
14. 加Amt1，Amt2，Amt3 的值到Total中.
15. Amt1中减去47. 5.

§3.8.3　实例

编写一个程序，计算汽车一加仑汽油行驶的英里数和每英里的成本. 下面的变量需输入:

BeginningMileage
EndingMileage
GallonOfGas
CostOfGas

对于每组输入变量，在打印机上打印出每加仑英里数和每英里的成本.

输出样例

MILEAGE REPORT

Miles Per Gallon	Cost Per Mile
12.5	.1199999
9.5	.1210526
14.46429	.1234568

第六章将详细地指出每英里的成本如何按美元和美分打印。为此，现在简单地给出 LPRINT USING 指令如下:

```
LPRINT USING "##.##"; CostPerMile
```

注意，打印的每英里成本可能有不同的四舍五处理，这取决于所用 Turbo Basic 的版本。

输入样例

```
124.00, 12500, 8, 12.C0
12950, 13045, 10, 11.50
13045, 13450, 28, 50.00
```

注意:

a. 输出是以MILEAGE REPORT为表头的报表。

b. 每加仑的英里数通过从EndingMileage 中减去BeginningMileage，然后再除以GallensOfGas 的计算得到。

c. CostPerMile可通过将CostOfGas除以EndingMileage 与BeginningMileage之差的方法得到。

§ 3.8.4　复习题

Ⅰ. 是非题

1. LPRINT在屏幕上显示输出，PRINT在打印机上打印输出.
2. 以 $ 结尾的变量名能够含有任何字母字符.
3. 常量有固定值，不管输入如何它始终是相同的.
4. 结构化程序设计使程序容易阅读，调试，维护和修改.
5. 变量名不能包含任何数字.
6. DO WHILE … LOOP循环的作用是重复一系列指令执行，直到指定条件满足为止.
7. 字符串变量不可以用在算术运算中.
8. 字符串变量名以%结尾.
9. 在算术运算中，乘法优先权高于除法.
10. 语句Total＝A+B×C 是C乘以A+B.

Ⅱ. 综合题

1. 指出下面哪些是字符串变量、数字变量、数字整形变量，或者错误变量名.

   a. ItemNo
   b. ItemNo%
   c. %ItemNo%
   d. Description $
   e. Item Description
   f. $ Description
   g. Cost
   h. 1Cost

2. 考虑下面的程序段:

```
LET A = 0
LET B = 7
LET C = 12
```

```
LET D = 25
LET A = D / 5
LET B = A + 2.75
LET C = B + 5 * A
LET D = C / 2 * A
```

在上面的指令执行完后，A，B，C，D 的值是什么。

3. 写一个Turbo Basic程序，在屏幕上显示初始值。
4. 虽然伪代码应在编程前写好，但仍请用户为本章末的实例写一个伪代码。
5. 指出下列常量中哪些是错误的?

   a. 1,450
   b. REPORT HEADING
   c. $500
   d. 1.765_
   e. "TOTAL DUE"
   f. +17.50
   g. 7 ^ 2
   h. 453.95

6. 写一个有合适标题的程序，打印出一年中的十二个月以及每月相应的天数。
7. 写一个Turbo Basic 程序，计算公司每个雇员的周薪 . 输入的是EmployeeName $，HoursWorked 和 HourlyPate. 假定所有雇员都工作在 40 小时以上，且都允许超进工作. OvertimeRate 是 HourlyRate 的 1.5 倍. 输出应是有列标题的报表. 报表应包含有 EMPLOYEE NAME（雇员名）和 WEEKLY PAY （周薪），并打印测试日期.
8. 何时宁愿让输出显示在屏幕上，而不是打印在打印机上?
9. 在Turbo Basic中，取一个合法的字符串和数字变量名的规则是什么?
10. 考虑下面的例子:

```
INPUT Amt1, Amt2, Amt3
```

响应 INPUT 语句时，若用户没有键入足够多的数据项，会怎样?

```
? 15,10    ←  然后按回车键
```

11. 输入TotalTuition，然后TutalTuition 除以15，得到TuitionPerCredil。
12. 将Amt1乘以3的值赋给Amt2。
13. 将Amt2增加1.
14. 找出X被Y除后的余数，并把它放在Z中.
15. 将Amt2减7.

§ 3.8.5 调试练习

下面的程序中若有错误，请指出来:

1. INPUT Amt of Sales
2. Nmae2$ = OtherName$ + 1
3. SUM = 0

```
PRINT "Enter The Amount, Type 0 To Stop"
INPUT Amount
DO WHILE Amount <> 0
      Sum = Sum + Amount
LOOP   PRINT "The Sum Is"; Sum
END
```

4. Count = 0

```
PRINT "Enter The Sales Amount, Type -999 To Stop"
INPUT Sale
DO WHILE Sale <> 0
     Count = Count + 1
     Total = Total + Count
     INPUT Sale
LOOP   PRINT "The Sum Is"; Sum
PRINT "The Sum Is"; Sum
END
```

5. PRINT "Type Height, Width"

```
PRINT "Type -0, 0  To End"
INPUT Height, Width1          ' Width 为保留字
DO WHILE Height Width1 <> 0
     PRINT "Area "; Width1 + Height
    INPUT Height, Width1
LOOP   END
```

6. Amt2 = Amt1 × 3
7. Product1 = 2Amt3

注意: 通常，调试练习没有答案。本章中含有答案，是为了让用户更好地理解每一问题。

§ 3.8.6 编程题

若要求交一份 Turbo Basic 编程作业，就需要得到一份程序和输出结果的硬拷贝。为了得到这样的输出，应按如下步骤进行：

1. 打开打印机。
2. 当程序显示在RUN窗口和输入及输出出现在运行窗口时，按下键盘左边的 Shift 或 ↑ 键；与之同时，按下键盘右边的 Prtsc 键，这叫打印屏幕。于是，整个屏幕就打印在打印机上。若没有图形打印机，或打印机没有配置图形模式，则边框可能以文本字符而不是线条打印，但编辑和运行窗口上的内容将按屏幕上的显示打印。
3. 若程序太长，编辑窗口放不下，即使放大编辑窗口也不行。在这种情况下，用上面 2 中的方法可以按屏幕显示分部分打印程序。
4. 最后一个选择是用常规方法保存文件。然后，按Alt-X返回DOS。按Ctrl 和 PrtSc 两键将引起回送打印。然后，键入：

   TYPE　<程序名>

   注意，程序名一定要包含扩展名·BAS。回送打印（Echo Prining）会使显示在屏幕上的程序也打印在打印机上。

下面的要求适合于所有编程题：

1. 假定每种情况输入行数是变化的。除非另有说明，否则使用结束值结束循环。
2. 用伪代码开始规划程序。
3. 运行程序时要有测试日期。
4. 当程序调试好后，打印程序和输入、输出的硬拷贝。若所有的都显示在主屏幕上，就按 Shift（↑），并且同时按下 PrtSc 键。这将回送打印（echo-print）屏幕上的一切。如果程序或运行不能全部放在主屏幕上。则放大每一个窗口。然后按 Shift-PrtSc 以得到一份硬拷贝。
5. 确保每个程序开始处含有注释。在提示输入时，要有清楚简洁的信息。

1. 输入：股票名称，每股价格，最后收入。
   打印：

股票名称　　每股价格　　最后收入　　P/E 比值

P/E 比值是每份收入的价格，是通过每股价格除以最后收入计算得到的。

2. 输入：姓名，毛收入，联邦税收率，州税率。

打印：

姓名　　毛收入　　联邦税　　州税　　净收入

联邦税 = 毛收入×联邦税率
州　税 = 毛收入×州税率
净收入 = 毛收入—联邦税-州税

3. 考虑下面的公式，计算销售价格：

$$销售价格 = \frac{1}{1 - 盈利率} \times 成本$$

编一个程序，读入每项的成本，然后计算销售价格；假定盈利率为 0.30。

4. 写一个程序打印下面的乘法表：

	One	Two	Three
One	1	2	3
Two	2	4	6
Three	3	6	9

5. 输入：在每行上输入学生名和每个学生的五门考试成绩。

打印：名字　　平均成绩

最后，当再没有输入时，打印出全班平均成绩。

第 二 部 分
逻辑控制结构细述

本部分内容

第四章　IF 选择语句
第五章　关于逻辑控制及自顶而下编程的迭代、循环和子程序

第四章　IF 选择语句

本章要求:

1. 熟悉使用IF选择语句块;
2. 能够根据不同的条件语句选用相应的格式.

§ 4.1　四种逻辑控制结构概述

§ 4.1.1　顺序结构

到目前为止，我所已经学习了一些 Turbo Basic 的基本语法规则. 从中可以看到，除非碰到 DO WHILE ... LOOP 语句，指令都是按顺序逐条执行的. 指令执行的方式就称作逻辑控制结构。

逻辑控制结构可以用于任何程序设计语言，其四种形式如下:

逻辑控制结构
1. 顺序结构(Sequence) 2. 选择结构(Selection)(IF−THEN−ELSE) 3. 迭代结构(Iteration) (DO WHILE ... LOOP or other Loop) 4. Case 结构(Case) (SELECT CASE)

顺序结构的伪代码为:

```
START
    :
    :  } 语句组
    :
STOP
```

例如:

```
' This program calculates a total
INPUT Amt1, Amt2
LET Total = Amt1 + Amt2
```

```
PRINT "The Total is "; Total
END
```

程序中没有其它的逻辑控制结构，指令从头到尾顺序执行．这种不论现场条件如何，指令都逐条顺序指行的方式，称作顺序结构．

§4.1.2 选择结构

如果程序中指令的执行取决于现场存在的一个或多个条件，这种方式被称作选择结构．它是一种单分支结构．有时也被称作 IF-THEN-ELSE 结构．因为它具有如下的伪代码形式：

```
IF (条件满足或"真") THEN
    :
    :  } 条件满足或"真"时要执行的指令
    :
ELSE
    :
    :  } 条件不满足或为"假"时要执行的指令
    :
ENDIF   ←  ENDIF 为 IF 结构的终止符
```

大多数程序设计语言都有类似于如上的 IF-THEN-ELSE 逻辑控制结构，本章将讨论 Turbo Basic 中的这种结构．下面简单介绍另外两种逻辑控制结构．

§4.1.3 迭代结构

迭代是一种能使一段指令反复执行的逻辑控制结构，到目前为止，我们已经用过 DO WHILE ... LOOP 语句进行迭代．下一章将全面讲述 Turbo Basic 中可以用于迭代的指令．

DO WHILE.....LOOP 结构的伪代码如下：

```
DO WHILE (条件存在)
    :
    :  } 要执行的程序段
    :
ENDDO
```

例如：

```
INPUT AmtIn
DO WHILE AmtIn < 99999
    PRINT AmtIn
    INPUT AmtIn
```

```
LOOP
END
```

§ 4.1.4 Case 结构

Case 结构是一种多分支结构，程序执行走向取决于一个变量的值。例如，在修改工资单时，如果变量的输入值为 1，意味着我们想变更雇员的薪水，如果变量的输入值为 2，则意味着变更的是雇员的名字，等等。可以用菜单的形式出现在显示器上。

```
                PAYROLL PROCEDURE MENU

Salary Change             [1]
Name Change               [2]
Address Change            [3]
Department Change         [4]
Type a Number 1-4         [ ]
```

用户只要输入 1、2、3 或 4，相应的程序就被保证执行，且完全取决于输入的数字。尽管也可以使用 IF 语句来判别输入的数字，但是 CASE 结构更简洁有效，伪代码形式如下:

伪代码

```
CASE 选单条目
    WHEN 1 处理 Salary Change(修改工资)
    WHEN 2 处理 Name Change(修改姓名)
    WHEN 3 处理 Address Change(修改地址)
    WHEN 4 处理 Department Change(改变部门)
    WHEN 0 打印错误信息
END CASE
```

第八章将介绍这种结构在 Turbo Basic 中可编码为:

程序节选

```
' An Example of a Case Structure
INPUT  MenuItem
SELECT  CASE  MenuItem
    CASE  1
             SalaryChangeProcedure
    CASE  2
             NameChangeProcedure
    CASE  3
             AddressChangeProcedure
    CASE  4
             DepartmentChangeProcedure
    CASE  ELSE
             PRINT "Menu  item  is  in  Error"
END  SELECT
```

其中每个命名的程序都是一个独立的子程序，一旦相应的条件满足，就执行与其对应的子程序，例如：若 MenuItem = 1，就执行称作 SalaryChangeProcedure 的子程序；若 MenuItem = 2，就执行称作 NameChangeProcedure 的子程序，以此类推.

§ 4.2 IF 选择语句块

本章重点讨论 IF-THEN-ELSE 选择结构，它能够根据变量的内容去执行一个或一系列的指令. IF-THEN-ELSE 结构在 Turbo Basic 中称作 IF 语句块.

§ 4.2.1 IF 语句块的指令格式

条件语句(Conditional statement)就是根据某些条件是否满足(或是否为真)来决定运行操作. 在 Turbo Basic 中，这类语句总是由 IF 开始，并被称作 IF-THEN-ELSE 语句块. Turbo Basic 支持简化的 If 和 If 语句块，这里着重介绍后者，因为它更具结构性，且和 IF-THEN-GO 结构更趋一致.

前面已经提到对应于 IF-THEN-ELSE 语句块选择结构的伪代码是:

伪代码

```
IF  <条件>  THEN
    ⋮  } 语句组
ELSE
    ⋮  } 语句组
ENDIF
```

注意：在伪代码中 ENDIF 是一个词。

IF 语句块的基本指令格式如下：

```
IF  <条件>  [,]  THEN
    ⋮  } 语句组
[ELSE
    ⋮  } 语句组]
END  IF
```

象伪代码中那样，在 Trubo Basic 程序中 IF 语句块也用 END IF 来结束，但是此时为两个分开的词．

在 IF 语句块中的表达式是用相应的运算符号组成的．其典型的结果是："真"或"假"．用于条件判断的关系运算符如下：

操作数 1	关系运算符	操作数 2	含　义
{数值1 或 变量1}	=	{数值2 或 变量2}	若操作数 1 等于操作数 2
	<		若操作数 1 小于操作数 2
	>		若操作数 1 大于操作数 2
	<=		若操作数 1 小于等于操作数 2
	>=		若操作数 1 大于等于操作数 2
	<>		若操作数 1 不等于操作数 2

条件语句中的操作数既可以是数值(常数)，也可以是变量．表达式中的操作数必须是

同一类型，也就是字符串变量或数值只能和其它的字符串变量或数值相比较。数字变量只能和其它的数字变量或数据相比较。

有效的关系型表达式
1. IF Tax = .25 THEN ......
2. IF Salary1 < > Salary2 THEN ......
3. IF Sales > = 500.00 THEN ......
4. IF EmpName$ = "Paul Newman" THEN ......

下列表达式是无效	无效的原因
1. IF Tax=-.25 THEN	1. Tax是数字变量，不能与".25"一个串常数比较。
2. IF IN < >In2$ THEN	2. In1是数字变量不能与In2$字符串变量比较。
3. IF Sales$ > =500.00 THEN	3. Sales$是字符串变量，不能与500.00数字常量比较。

在 Turbo Basic 中，把关系运算符及其联接起来的操作数合称作条件表达式。IF 语句的结果将取决于其条件表达式是"真"或是"假"，参看下面的 IF 语句块：

```
IF  Hours > 40 THEN
    PRINT "Overtime Pay Required"
ELSE
   PRINT "Regular Pay Only"
END IF
```

上面语句的判断结果有两种可能：

(a) 大于 40 小时　　　　条件满足或为"真"
(b) 不多于 40 小时　　　条件不满足或为"假"

说明：

(a) 如果大于40小时，数据"over time pay required"被打印，跳过ELSE后面的部分，直接执行 END IF 后面的指令，而不管 ELSE 和 END IF 之间的语句。

(b) 如果不大于40小时，则不执行第一个打印指令，只执行ELSE后面的语句，即

"regular pay only"，并顺序执行后面的语句。这样，利用 IF 语句块，可判断如果原始条件满足或是"真"，则执行指定的指令，利用 ELSE，当条件不满足或是"假"，则可以完成其它的操作。我们尽量使用缩格书写格式，IF 与 ELSE 之间的语句缩入数格，以便于阅读和调试。

我们也极力建议读者这样做。

§4.2.1.1 指令格式注释

A. THEN必须和IF分句在同一行上。

下述表达是错误的:

```
IF  Amt1 = 0
    THEN Total1 = Total1 + 1
ELSE
    Total2 = Total2 + 1
END IF
```

正确的表达式应该为:

```
IF  Amt1 = 0 THEN
    Total1 = Total1 + 1
ELSE
    Total2 = Total2 + 1
END IF
```

此外，在 IF 语句块中第一行的 THEN 后面不能跟其它的输入量。

B. 格式中括号内的逗号是可有可无的。

C. 格式中[ ]内的ELSE语句可有可无。

某些操作只要求在条件满足时执行，而不满足时则无其它操作，所以ELSE分句可以省略，但在任何情况下，IF 语句块必须以 END IF 语句结束。

无 ELSE 的 IF 语句用例

程序节选

```
IF  AmtOftrans  =  THEN
       PRINT "No Transaction This Month"
END  IF
```

伪代码

```
IF  AmtOftrans 为零  THEN
      PRINT  一条信息
END  IF
```

在这个例子中，只有当AmtOfTrans为零，信息"No Transactions This Month"才被打印出来。假如 AmtOfTrans 不为零，则执行 END IF 以后的指令而不进行任何其它操作。由于 AmtOfTrans 不为零时不需要任何操作，则 ELSE 分句在这个例子中是不必要的。

D. 格式中的"语句组"表示在条件为"真"(或条件满足)时，可进行一种以上的操作；同样，在条件为"假"(或不满足)时，也可进行一种以上的操作。

为了便于阅读，一般把每个要执行的语句单写一行，并缩进。同样，ELSE后的语句也要缩进。因此，下述语句在 Weight 大于 200 时要执行两个操作，在条件不满足时也要执行两个操作。

程序节选

```
IF  Weight  >  200  THEN
      PRINT  "Think About a diet"
      OverWeightCtr  =  OverWeightCtr  +  1
ELSE
      PRINT  "Your Weight is OK "
      OkWeightCtr  =  OkWeightCtr  +  1
END  IF
```

伪代码

```
IF  体重超过 200  THEN
      PRINT  一条信息
      OverWeightCtr 加 1
ELSE
      PRINT  一条信息
      OkWeightCtr 加 1
END  IF
```

类似地，在下面的语句中，无论字符串变量等于"M"与否，也将分别执行两个操作。

程序节选

```
IF Sex$ = "M" THEN
    PRINT "Sex is Male "
    MaleCtr = MaleCtr + 1
ELSE
    PRINT "Sex is Female "
    FemaleCtr = FemaleCtr + 1
END IF
```

伪代码

```
IF 性别码为"男" THEN
    PRINT 一条信息
    MaleCtr 加 1
ELSE
    PRINT 一条信息
    FemaleCtr 加 1
END IF
```

§ 4.2.1.2 同类操作数比较和数字操作数与算术表达式的比较

我们已经学过了数字变量与数字常量，数字变量与数字变量，字符串变量与字符串变量，字符串变量与字符串常量之间的比较；还要指出的是，数字变量也能与算术表达式进行比较。

例:

```
' QtyOrdered is compared to the sum of QtyOnHand+100
IF QtyOrdered > QtyOnHand+100 THEN
    PRINT "Quota Exceeded"
END IF
```

如果混淆了数据类型会发生什么呢?

如前所述，为获得正确的结果，条件表达式必须使用同一类型的变量．考虑下面的非法用例:

```
INPUT  Code$                              ' Code$ 为字符串变量
IF  Code$  <  0  THEN   ' 0 为数值常数
    PRINT  "Negative Code"
ELSE
    PRINT  "Positive Code"
END  IF
```

当计算机编译这个程序段时，指针会自动指到另并在屏幕上显示如下语法错误．

```
―――――――――――――――――― Edit ――――――――――――――――――
Error 419: String expression requires string operand
```

这是由于 Code$ 是一个字符串变量，而-0″是一个常数，因此，必须谨记，在一个关系表达式中两个操作数必须是同一类型．

§ 4.2.1.3 编码指导

前面已经举例说明了如何编写 IF 语句块，本节将正式给出整个结构的解释．

A. IF 块的每个语句写成一行

一个 IF 块应该写成如下形式:

```
IF  <条件>[, ]THEN    ←  这条语句单独占一行
    语句 1
    语句2 }  建议一行只写一条语句，并缩格.
      :
ELSE    ←   单独占一行
    语句 3
    语句4 }  建议一行只写一条语句，并缩格.
      :
END IF    ←   单独占一行
```

一个 IF 块首行必须也只能写成: IF <表达式> THEN 形式．其后是条件满足时，要执行的语句．每个语句各占一行; ″ELSE″必须独占一行，随后是条件不满足时要执行

的语句，每个语句各占一行；最后一行为"END IF"。

一个 IF 语句块也能写在下面的形式。

```
IF  <条件>[，]THEN
      语句 1 : 语句 2            ' 各语句间用冒号分隔
ELSE
      语句 3 : 语句 4            ' 各语句间用冒号分隔
END IF
```

若几个语句写成一行，中间要用冒号(:)分开，但相比之下，后者更难阅读，也不利于调试。

B. 语句缩进

在 IF 块中，除 IF ...THEN 、ELSE、END IF 三行外，其余各行均缩入数格，写成如下形式：

```
IF  <条件>  THEN
     语句1 ┐
     语句2 ├ IF 之后的所有语句均缩格。
       ⋮   ┘
ELSE
     语句3 ┐
     语句4 ├ ELSE 之后的所有语句均缩格。
       ⋮   ┘
END  IF
```

这种标准的编写格式，程序结构显而易见，便于阅读和调试，但是无论采用哪种格式，并不影响编译处理，也就是这种编写技术不必须的。读者可用标有"⇌"的 Tab 键缩进 8 个空格。

下面的编写方式每行写有多个语句。

```
IF Amt1 = Amt2 THEN
      Tot = Amt1 / Amt2 : Ctr1 = Ctr1 + 1
ELSE
      Tot = Tot + 10 : Ctr1 = Ctr1 + 2
END IF
```

如果改写成下面的格式将更利于阅读和调试：

```
IF Amt1 = Amt2 THEN
```

```
        Tot  =  Amt1 / Amt2
        Ctrl  =  Ctrl + 1
ELSE
        Tot  =  Tot + 10
        Ctrl  =  Ctrl + 2
END IF
```

例题一

下面的伪代码要求输入销售量并计算、打印佣金。如果销售量小于 500.00，佣金是 5%，否则佣金是 10%。

伪代码

```
START
    输入销售量
    DO WHILE 还有输入?
        IF 销售量小于$500.00 THEN
            佣金为 5%
        ELSE
            佣金为 10%
        END IF
        打印一条信息
        输入销售量
    ENDDO
STOP
```

Turbo Basic 程序

```
'This program calculates and prints commission
CLS
PRINT "Enter Sales Amount - Type 99999 When Done"
INPUT SalesAmount
DO  WHILE SalesAmount < 99999
    IF  SalesAmount < 500 THEN
        Commission = .05 * SalesAmount
    ELSE
        Commission = .10 * SalesAmount
    END IF
    PRINT "Commission is "; Commission
    INPUT SalesAmount
LOOP
END
```

例题二

输入某公司全体职工的姓名和工资，计算并打印男性和女性职工的平均工资。

伪代码

```
START
    初始化职工人数计数器和工资总额
    输入性别和工资
    DO  WHILE  还有输入?
        IF  性别为“女”  THEN
            女性职工的计数器加一
            将输入的工资加入女性职工的工资总额
        ELSE
            男性职工的计数器加一
            将输入的工资加入男性职工的工资总额
        END  IF
        输入性别和工资
    ENDDO
    计算并打印女性职工的平均工资。
    计算并打印男性职工的平均工资。
STOP
```

打印男性和女性职工平均工资的完整程序如下:

EDIT 窗口

```
'This program calculates average salaries for males and females
CLS
FemaleCtr = 0
MaleCtr = 0
TotalFemaleSalaries = 0
TotalMaleSalaries = 0
PRINT "Enter Sex (M,F) and Salary"
PRINT "Enter 9,999999 when done"
INPUT Sex$, Salary
DO WHILE Sex$ <> "9"
       IF Sex$ = "F" THEN
              FemaleCtr = FemaleCtr + 1
              TotalFemaleSalaries = TotalFemaleSalaries + Salary
       ELSE
```

```
            MaleCtr = MaleCtr + 1
            TotalMaleSalaries = TotalMaleSalaries + Salary
        END IF
        INPUT Sex$, Salary
LOOP
AverageFemaleSalary = TotalFemaleSalaries / FemaleCtr
AverageMaleSalary = TotalMaleSalaries / MaleCtr
PRINT "The Average Salary for Females is ";AverageFemaleSalary
PRINT "The Average Salary for Males is ";AverageMaleSalary
END
```

RUN 窗口

```
Enter Sex (M,F) and Salary
Enter 9,999999 when done
? M,1000
? F,500
? F,500
? M,1000
? 9,999999
The Average Salary for Females is  500
The Average Salary for Males is  1000
```

自我测验(一)

首先写出下列各题的伪代码，并编出相应的程序.

1. 查阅雇员名单和应得工资，如果雇员所得工资少于5万美元($ 50000)，联邦所得税为所得工资的 20%，否则联邦所得税为 25%，计算并打印每个雇员的名字和应交的所得税.
2. 编程计算为收入少于或等于 $ 25000的雇员发放奖金10%. 为收入多于 $ 25000 的雇员发放 8%，打印每个雇员的名字及奖金额，及公司要付的奖金总额 .

§ 4.2.2 比较的规则

§ 4.2.2.1 数字量比较

在数字变量的比较中，下面的几个数是相等的.

012　12.00　12　+12

也就是，Turbo Basic 中的数字比较采用代数规则．被比较的数字操作量的位数并不影响比较结果．尽管每个数的位数不同，但，001＝01＝1；此外，正数等于无符号的数．

§4.2.2.2　字符串操作数的比较

在字符串操作数的比较中：

A ＜ B ＜ ..... ＜ Z
a ＜ b ＜ ... ＜ z

A．字符串变量中左侧空格的影响

在赋值语句中已经赋值的字符串变量比较时，下列的写法是不等的．

ABC，b̸ABC，b̸b̸ABC
（b̸表示一个空格）

带有高位或左侧空格的字符串变量被认为小于任何无空格的字符串变量，因此影响比较结果，请看下面的例子．

```
Constant1$ = "ABC"
Constant2$ = "  ABC"
IF  Constant1$ = Constant2$ THEN
    PRINT "They are equal"
ELSE
    PRINT "They are not equal"
END IF
```

执行这段程序后，屏幕上将打出"They are not equal"，因为，"b̸ABC"（b̸为空格），被认为小于"ABC"．

如果常数或字符串带有高位空格只会影响等式的均衡．联想到不带引号的字符串变量，不包含高位空格，即使操作者在输入其它字符前按了空格键（Space）．请看下例：

Edit	Run
INPUT Constant3$	? ABC

←— 注意：应先按两个空格，再键入"ABC"．

在此，Constant3 $ 被存作"ABC"与上面的 Constant1 相等，尽管它带有两个前导空格．这样，在程序中，一个字符串变量带有前导空格唯一可能是：要么，赋值语句中包插

前导空格；要么，输入是带有引号" "的。意即，"bbABC"(b为空格)。总的来说，前导空格将影响比较结果。

B. 字符串变量中右侧空格的影响

如果两上变量包含相同的字符，但一个有右侧空格，一个无空格，则前者大于后者，例如:

```
IF "ABC" < "ABC  " THEN
    PRINT "Shorter string variables compare <"
END IF
```

信息: "Shorter string variables compare <" 将显示在屏幕上。字符串后面的输入空格影响比较结果，假如String1 $ 是"ABCbb"，(b为空格)或者根本就没有引号，ABCbb，在与"ABC"相比时，前者大于后者。

```
INPUT String1 $                ' 将"bbABC"输入 String1 $
IF "ABC" < String1 $ THEN
    PRINT "Shorter string variables compare <"
END IF
```

这里信息: "Shorter string variables compare <" 将再次显示出来。当ABCbb，(b为空格)被输入，计算机认为该字符串是 ABCbb，这是不同于前面讲到的前导空格。在输入 bbABC 时，计算机认为该字符串是 ABC，除非用引号把 bbABC 引起来，即"bbABC"。

C. 字符串操作数与空串的比较

在提示输入字符串变量时，如果操作者没有输入任何数据而直接按下回车键(Inter)，计算机认为该字符串变量是个空串，它是真正无值的，小于任何字符串变量，即使是一个空格。

```
INPUT A$                  ' 将"bbABC"输入 A$
IF "ABC" < "" THEN        ' A$与空串比较
    PRINT "Is NULL"       ' 打印"Is NULL"
END IF
```

一个空串可以通过直接按下回车键(Enter)获得，也可以被定义为 A $ =""；引号""内没有空格。

D. 总的来说，字符串变量中的字符数总影响比较结果

如前所述，数字操作数中的字符数并不影响比较结果；例如：

012 = 0012 = 12.0

但是在字符串比较中，当两个字符串变量前面的字符都相同时，串的长度就是一个关键的因素。假如两个不等长度的字符串变量相比较，比较直至较短串的最后一个字符时二者均相同；此时，就认为短串小于长串，而不论它长串余下的字符是什么。例如：

字符串数据比较	
	解　　释：
"DOG"<"DOGS" "DOG"<"DOG"(b为空格) "DOGb"<"DOGbb"(b为空格) "tes"<"test"	短字符串前面的字符与长字符串前面的字符相同时，短字符串小于长字符串。

> **规　　则**
>
> 在不同长度的字符串操作数比较中，参与比较的字符数目由较短的字符串决定。

E. 字符串变量比较不采用代数规则

应当注意，字符串操作量比较并不象数字操作量那样采用代数规则。在串比较中，"123"大于"0123"，因为前者高位的"1"大于后者高位的"0"。但是在数字变量的比较中，123 等于 0123。

§4.2.2.3　如果操作数 1 不大于等于操作数 2，那么它必小于操作数 2

下面的两段程序具有相同的结果

```
a. IF Amt1 >= 5 THEN
       Count1 = 0
   ELSE
```

```
        Count2  =  0
    END IF

b.  IF Amt1  <  5  THEN
        Count2  =  0
    ELSE
        Count1  =  0
    END IF
```

在(a)中，如果 Amt1 大于等于 5(Amt1≥5)，则执行 ELSE 语句且 Count2 被置零。在(b)中，如果 Amt1 小于 5(Amt1 < 5)，则执行语句 Count2 = 0，所以(a)和(b)是等效的。

§ 4.2.3 ASCII 码排序序列

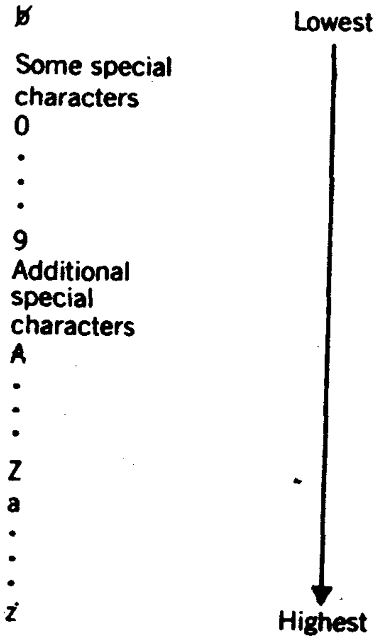
进行字母数字的比较时，比较的顺序称作"排序序列"，由计算机确定。

ASCII 码(American Standard Code for Information Interchange 之缩写)，是一种广泛用于数据描述的计算机内部编码。

在 ASCII 码中，字符的比较顺序如下：

ASCII 码的大小顺序

(自上而下，ASCII 码由小到大)



在采用 ASCII 码的计算机中，数字、大小写字母的比较将按上图严格执行。例如数字 012 小于 022，小于 042，因为：A < B < C ..... < Z，这样就有"ABCD"小于

"BBCD"，小于"XBCD"；类似地，a < b < c < … < z. 值得注意，在 ASCII 码计算机中，大写字母小于小写字母，即:

```
A < a
Z < z
TEX < tex
```

所以在输入数据或给一个字符串变量赋值时，如果混用了大写和小写字母，将得不到预期的比较结果.

大小写字母混用时的字符串比较

```
"ABC" < "abc"
"Abc" < "abc"
"ZIP" < "apple"
"ZIP" < "ZIp"
```

涉及字符数据时的编程准则

在输入字符串变量值或使用赋值语句给一个字符串变量赋值时，努力养成这样的习惯；或者都用大写字母或者都用小写字母；也可以把首字母大写. 用这种方法，可将字符串变量比较中的错误降到最低程度. 后面还会学到，也可使用 UCASE $ 或者 LCASE $ 函数，来指定字符串都为大写，或都为小写. 这些函数可以使比较不受影响.

再有一点，在计算机中，数字的 ASCII 码小于字母的 ASCII 码. 例如在一个地址变量中"Route 109"大于"100 mains"，因为字母"R"大于"1".

§ 4.2.4 条件满足时不操作

§ 4.2.4.1 IF<条件>THEN ELSE … END IF

有时会遇到这样的情况，只需要在条件不满足或为"假"时，执行一系列的操作.

例 1 考虑下面的伪代码:

伪代码

```
START
```

```
        ⋮
        IF  Amt1  =  Amt2  THEN
             下一条语句
        ELSE
             将总数加 1
        END  IF
        ⋮
    STOP
```

在 Amt1＝Amt2 时，无操作，也就是我们想要计算机继续执行 END IF 后面的语句。如果 Amt1≠Amt2，则总数(Total)加 1，为此伪代码对应的 Turbo Basic 程序段是:

```
IF  Amt1  =  Amt2  THEN
ELSE
     Total  =  Total  +  1
END  IF
```

注意，在 IF.....THEN 和 ELSE 之间没有任何语句，在条件满足时，若无操作，就可略去不写。

§4.2.4.2 使用 EXIT IF 跳出 IF 语句块

在条件满足时，EXIT 语句也可用来避免进行进一步的处理。以后，我们还会看到一个 EXIT 语句可以是选择、迭代、或多分支(Case)逻辑控制结构的一部分。在错误发生时，往往用它来及时跳出结构。

使用 EXIT IF 指令，上例可该写成如下形式:

```
IF  Amt1  =  Amt2  THEN
     EXIT  IF        ' 若 Amt1  <>  Amt2，则退出 IF 块
ELSE
     Total  =  Total  +  1
END  IF
```

总的说来，EXIT 语句被广泛地用于逻辑控制结构中，用以在错误状态发生时，避免处理一系列的步骤。

自我测验(二)

在 1~3 题中，存在什么错误?

1.
```
IF  Amt  =  0
        PRINT  "Amt Is Zero"
    END  IF
```

2.
```
IF  Amt  =  "127"  THEN
        LET  Total  =  Total  +  1
    END  IF
```

3.
```
IF  Amt1  =  Amt2  THEN
        Counter  =  0
    ELSE
        Counter  =  100
```

4. 编程让变量PrintSmall等于A、B、C三个变量中的最小一个.
5. 如果两个串具有相同的长度，它们将逐位进行比较(T或F)

指出 6~11 题中两个字符串比较的结果.

6. "Adams"　　"Baker"
7. "IBM"　　"ibm"
8. "Turbo"　　"Turbo Basic"
9. 123　　0123
10. "123"　　123
11. ""　　" "
12. 输入某人的姓名、工作时数和编码，如果编码为1，单位时间的报酬为 5.00，若编码为 2，单位时间的报酬为 10.00,首先写出伪代码，然后编出 Turbo Basic 程序.

§4.3　IF 语句的其它选择方式

§4.3.1　IF 语句块与标准 IF 语句的对比

与 IF 块相比，一个标准的 IF 语句格式如下:

```
IF  <条件>[, ] THEN  <语句>  [ELSE  <语句>]
```

在这种情况下，IF 语句和 ELSE 后的相关语句写在同一逻辑行上，语句之间用冒号(:)隔开。如果一个自然行写不下，必须用连接符(_)连起来。一个逻辑行可以包括任意多个自然行。

举例:

```
1. IF Total > 1000 THEN PRINT "Discount Available" ELSE PRINT_
     "No Discount"
2. IF Amount1 > 1000 THEN PRINT "Amount is High"_
   ELSE PRINT "Amount is Low"
3. IF Sex$ = "M" THEN MaleCtr = MaleCtr + 1: PRINT "Male"_
   ELSE FemaleCtr = FemaleCtr + 1: PRINT "Female"
```

> **注意:** 一般认为，标准IF语句难以阅读和调试，因而不推荐使用。相反，我们推荐大家使用 IF 语句块。

§ 4.3.2 条件句嵌套

条件句嵌套就是在 IF 语句中又包括新的 IF 分句，参看下面举例:

§ 4.3.2.1. IF ... THEN IF ... END IF 嵌套

例 1

(注: 本例中的行号仅仅是为了解释的方便。)

```
10  IF  Dept = 1 THEN
20       IF  Hours > 40 THEN
30           Bonus = 3 * Hours
40       ELSE
50           Bonus = 2 * Hours
60       END IF
70  ELSE
80       Bonus = 0
90  END IF
```

这个例子写成了我们最早描述的 IF 结构形式，不同的是: 如果第一个条件为"真"，还要进行另一个测试。这就是一种条件句嵌套。

程序执行过程是:

1. 如果Dipt≠1，则执行后一个即为70行的ELSE，并且将"Bonus置0。
2. 如果Dept=1，则执行第20行的第二个条件测试:

(a) 如果Hours 大于40，(Hour>40)，则Bonus=3 * Hours;

(b) 如 果 Hours 不 大 于 40， 则 执 行 40行 的 ELSE语 句 Bours=2 * Hours。

上例中各子句的缩格有助于嵌套条件结构的解释。

这个程序的伪代码可以写成如下判定表(Decision Table)。

Decision Table for Example 1		
Condition 1	**Condition 2**	**Action**
Dept = 1	Hours > 40	Bonus = 3 * Hours
Dept = 1	Hours <= 40	Bonus = 2 * Hours
Dept ≠ 1	Hours = anything	Bonus = 0

判定表列出了程序中可能发生的各种情况(Condition)和要执行的相应动作(Action)。程序员用它画出由不同测试结果来决定执行相应语句的复杂的逻辑关系。

一个条件句嵌套是一系列单一条件句的简写方式，我们将会看到，任何条件句嵌套都可以写成简单条件句的形式。

为什么要使用条件句嵌套呢?

第一，在多个条件测试时，它能缩短程序;
第二，它把条件检测写成就象判定表一样;
第三，它使程序的逻辑结构更简洁。

例 2 考虑如下判定表:

Condition 1	Condition 2	Action
Sex = "M"	Salary > 100000	PRINT "Sounds like a good catch"
Sex = "M"	Salary <= 100000	PRINT "He doesn't earn six figures"
Sex ≠ "M"	Salary = anything	PRINT "Not a male--ineligible"

用条件句嵌套方式编写该判定表。为了帮助弄清各语句的关系，采用如下缩格行式:

```
IF  Sex = "M" THEN
    IF  Salary > 100000 THEN
        PRINT "Sounds like a good catch"
    ELSE
        PRINT "He doesn't earn six figures"
    END IF
ELSE
    PRINT "Not a male--ineligible"
END IF
```

§ 4.3.2.2 IF ... THEN ... ELSEIF ... END IF 嵌套

条件句嵌套的另一种格式为:

```
IF  <条件 1>  THEN
      语句 1
       :
ELSEIF <条件 2> THEN
      语句 2
       :
ELSE
      语句 3
       :
END IF
```

例

```
IF  Salary > 99999 THEN
    PRINT "Six Figure Salary"
ELSEIF Salary > 60000 THEN
    PRINT "Salary Between 60000 and 99999"
ELSE
    PRINT "Salary Less Than or Equal to 60000"
END IF
```

注意，象 IF 一样，ELSEIF 后也跟一个条件，因此该条件后面也必须带有 THEN；此外，ELSEIF 是一个字。

该嵌套格式只有一个 END IF 。在一个块中，IF 总要与 ENDIF 配合使用的，但可有多个 ELSEIF。

自我测验(三)

1. 下面程序的输出是什么？

```
a. I = 10      b. I = 60      c. I = 20
   J = 20         J = 40         J = 30
   K = 30         K = 30         K = 10

IF  I < J THEN
    IF  I < K THEN
        PRINT I
    ELSE
        PRINT K
    END IF
ELSE
    IF  J > K THEN
        PRINT J
    ELSE
        PRINT K
    END IF
END IF
```

2. 编写一个程序计算并显示快信的邮资：第一盎司或不足一盎司的邮件,收25分，多于一盎司的部分每盎司收 17 分。输入盎司数，并计算邮资；使用条件句嵌套，并用 DO WHILE ... LOOP 输入数值。
3. 某销售VCRO的商店大酬宾，用标签的颜色指示商品的折扣率：

颜色	编码	折扣率
红	R	30%
绿	G	25%
兰	B	20%

使用条件句嵌套编写一个程序，输入颜色编码和价格和价格表，然后显示销售价格。若编码不是 R.G.B，请打印"No discount"。
4. 利用IF 语句块和条件句嵌套编程。输入StudentNane(学生姓名)和NoOfCredits(学分)。如果 NoOfCredits<12，学费是 $ 200 × NoOfCredits；如果 NoOfCredits 是 13～15，学费为 $ 2500；如果 NoOfCredits 是 16～18，学费为 $ 3200；如果 NoOfCredits 超过 18，请打印错误信息。
5. 用IF-THEN-ELSE语句嵌套编程。
某个营业员其收入为工资与佣金之和，如果销售额超过 $ 5000，佣金是8%；如果销售额不超过 $ 5000，佣金为 5%。这些适用于工资超过 $ 1000 的人，如果工资不超过 $ 1000，则佣金是 6.5%，而不计其销售额。计算并打印每人的佣金。

§ 4.3.3 复合条件语句

读者已经看到，选择结构为程序逻辑的控制提供了方便。复合条件句则为选择提供了更大的灵活性，使 IF 语句可以用于更复杂的问题。使用复合条件句程序员能够在一个 IF-THEN-ELSE 结构中测试几个条件。

§ 4.3.3.1 OR 复合条件句

如果在数个条件中的任意一个满足时，进行一个或一系列操作，就需要使用一个复合条件句，每个条件用 OR 分开，这意味着如果数个条件中的任意一个满足，则执行相应的语句。

复合条件句的格式如下：

```
IF  <条件1>  OR  <条件2>  ... THEN
      :
      :  } 语句组
      :
[ELSEIF  <条件3> [,..]  THEN
      :
      :  } 语句组]
      :
[ELSE
      :
      :  } 语句组]
      :
END IF
```

注意：在所有的条件句中，ELSEIF 和 ELSE 分句是可有可无的。

例 1

```
IF Amt1  <  Amt3  OR  Amt1  =  Amt4  THEN
     LET  Total  =  Total + Amt1
ELSE
     LET  Total  =  0
END  IF
```

例 2

```
IF  Sex$  =  "M"  OR  Sex$  =  "F"  THEN
     PRINT  "Sex Code Is Correct"
```

```
END IF
```

复合条件句中的连接词 OR 使得指定条件句中任一个都可导致同一语句的执行．如果所有条件都不满足，计算机将执行 ELSE 或 ELSE IF 分句(如果有的话)，或者 END-IF 后面的语句．在一个语句中可以有包括用 OR 连接起来的任意多个条件．

在例 1 中，只有当 Amt1>Amt3，并且 Amt1≠Amt4 时，Total 等于 0(Total =0)，如果 Amt1<Amt3 或者 Amt1=Amt4 之一成立，则 Total=Total+Amt1，然后计算机继续执行 END IF 后面的语句．

在例 2 中，请记住与字符串变量比较的字符串常量必须用引号(" ")，此外，在每一个条件中必须是同一类型的操作数．但是用 OR 连接起来的条件中可以有串比较或数字比较．请看下例:

```
IF Dept = 1 OR Location$ = "New York" THEN
    PRINT "Attend thi meeting"
END IF
```

§4.3.3.2 复合条件句的限制

A. 标志复合条件句开始的IF只能使用一次

下述写法是错误的:

```
IF Amt1 ≟ 0 OR IF Amt2 = 0 THEN
    PRINT "Error"
END IF
```

这种类型的复合条件句 IF 只能用一次

B. 避免使用隐含操作数

有时，我们用"IF Amt1 OR Amt2=0 THEN ... "来表示"IF Amt1=0 OR Amt2= 0 THEN ... "．前者 0 只出现在 Amt2 的关系式中，故被称为隐含操作数；类似地，用"IF Sex$ ="M" OR "F" THEN ... "代替"IF Sex$ ="m" OR Sex$ ="F" THEN ... "也被称作隐含操作数．在有些场合，计算机能够接受隐含操作数，但有时就不能．因此，我们坚决要求避免使用，以减少错误的发生．

等效的比较形式:

有时复合条件可以写成简单条件句，例如:

```
IF Amt1 < 5 OR Amt1 = 5 THEN.....
```

就可写成:

```
IF Amt1 < 5 THEN ...
```

下面三个程序其结果是相同的.

```
1. IF Amt1 <= 100 THEN
        PRINT "Amt1 is small"
   ELSE
        PRINT "Amt1 is large"
   END IF

2. IF Amt1 < 100 OR Amt1 = 100 THEN
        PRINT "Amt1 is small"
   ELSE
        PRINT "Amt1 is large"
   END IF

3. IF Amt1 > 100 THEN
        PRINT "Amt1 is large"
   ELSE
        PRINT "Amt1 is small"
   END IF
```

§ 4.3.3.3 AND 复合条件句

如果只有在几个条件都满足时，才执行某些语句，可以在复合条件句中使用 AND 来实现，其格式为:

```
IF <条件 1> AND <条件 2> ... THEN
   :
   : } 语句组
   :
[ELSEIF <条件 3> [, ] THEN
   :
   : } 语句组]
   :
[ELSE
   :
   : } 语句组]
   :
END IF
```

复合条件句中用 AND 连接诸条件时，所有条件必须同时满足；否则，若其中任一个不满足，计算机将执行 ELSE 或 ELSEIF 分支(如果有的话).

例1 假定在条件:

Sex$ = "M"、高度小于 72 英寸，重量小于 200 磅

都满足时，打印"This is a male and six feet and < 200 lbs"；否则，打印"This person does not meet all three conditions"。也就是说，在一个或多个条件不满足时打印后者．用复合条件句写出的程序为:

```
IF  Sex$ = "M" AND Height < 72 AND Weight < 200 THEN
    PRINT "This is a male under 6 ft and < 200 lbs"
ELSE
    PRINT "This person does not meet all three conditions"
END IF
```

例 2. 假定输入推销员姓名和销售额，并按如下方式计算佣金:

A) 如果销售额小于等于 500，佣金为 2%.
B) 如果销售额大于 500 小于 1000，佣金为 4%.
C) 如果销售额大于等于 1000，佣金为 6%，打印推销员姓名和佣金.

伪代码

```
START
    输入推销员姓名和销售额
    DO WHILE 还有输入?
        IF 销售额 <= 500 THEN
            佣金为销售额的 2%
        END IF
        IF 销售额大于 500 或小于 1000 THEN
            佣金为销售额的 4%
        END IF
        IF 销售额大于或等于 1000 THEN
            佣金为销售额的 6%
        END IF
        打印佣金
        输入推销员姓名和销售额
    ENDDO
```

STOP

程 序

```
CLS
PRINT "Enter Salesperson Name and Sales Amount"
PRINT "Type End, 99999 when done"
INPUT SName$, Amt
DO  WHILE Amt < 99999
    IF  Amt <= 500 THEN
        Commission = .02 * Amt
END IF
IF  Amt > 500 AND Amt < 1000 THEN
    Commission = .04 * Amt
END IF
IF  Amt >= 1000 THEN
    Commission = .06 * Amt
END IF
PRINT SName$; "earned "; Commission
INPUT SName$, Amt
LOOP
END
```

§4.3.3.4 AND 和 OR 用于同一语句

在一个复合条件句中，可以同时使用 AND 和 OR:

例 1:

```
INPUT Sex$, Age
IF  Sex$ = "F" OR Sex$ = "M" AND Age > 25 THEN
    PRINT "Regular Insuree"
ELSE
    PRINT "Assigned Risk"
END IF
```

复合条件句中 AND 和 OR 的计算顺序

在同时使用 AND 和 OR 的复合条件句中，如上面给出的例子中，计算顺序是怎样的呢? 各条件的计算顺序是至关重要的.

例 1 中可能的计算顺序: I

```
            (a) Sex$ = "F" OR Sex$ = "M"
     AND    (b) Age > 25
```

在这种情况下，"Regular Insuree" 只有在(a)和(b)同时成立时才打印。即，不分男女只要大于 25 岁就被认作"Regular Insuree"，而小于或等 25 岁的人，被认做"Assigned Risk"。

例 1 中可能的计算顺序：Ⅱ

```
            (a) Sex$ ="F"
     OR     (b) Sex$ ="M"   AND   Age>25
```

用这种计算顺序，"Regular Insuree" 在(a)、(b)之中任一个满足时都打印；即：女人和大于 25 岁的男人被认作："Regular Insuree"，也就是只有小于或等于 25 岁的男人才被认作"Assigned Risk"。

Turbo Basic 使用的计算顺序为：

> **分级规则**
>
> 第一，计算由 AND 连接的条件；
> 第二，计算由 OR 连接的条件；
> 第三，有多个AND和OR 时，首先自左至右的计算由AND连接的条件，然后是 OR 连接的条件。
> 第四，带有括号的条件最先计算。

利用这些分级规则，例 1 的计算顺序为：

```
            (a) Sex$ ="M"  AND  Age >25
     OR     (b) Sex$ ="F"
```

取而代之的是，将满足：Sex$ ="M" AND Age< =25 的人归入"Assigned Risk"。这种情况与上面所述的第二条计算顺序相同。

若想改变计算顺序，可以用加括号的方法来实现。

例 2

```
INPUT Sex$, Salary, Assets
IF Sex$ = "M" AND Salary > 100000 OR Assets > 250000 THEN
     PRINT "Male Yuppie"
```

由于没有括号，条件的计算顺序为:

```
         (a)  Sex$ = "M"  AND  Salary > 100000
OR       (b)  Assets > 250000
```

可以看出，即使是女士，只要有多于250000的资产也被认作"Male Yuppie"。为了得到正确计算顺序必须使用括号

```
INPUT Sex$, Salary, Assets
IF Sex$ = "M" AND (Salary > 100000 OR Assets > 250000) THEN
    PRINT "Male Yuppie"
```

此时的计算顺序变为:

```
         (a)  Assets > 250000 OR Salary > 100000
AND      (b)  Sex$ = "M"
```

这个计算顺序被称作布尔优先级。我们在后面的反条件句中还会讨论。
请注意，性别编码必须用大写的"M"或"F"程序才能正常运行。

在复合条件句用 AND 和 OR 编程的原则

为了简化编程，减少错误，在使用带有 AND 和 OR 的复合条件句时，最好使用括号。

判别变量值在某区间内的方法

请看下例，变量 Amt1 的值在[10, 13]内时，打印"It fits!"；变量 Amt1 的值等于 10 或 13 时，打印"Inclusive of the end points!"。

错误的解法一:

```
IF Amt1 = 10 OR Amt1 = 11 OR Amt1 = 12 OR Amt1 = 13 THEN
    PRINT "It fits!"
END IF
```

只有在Amt1是整数时，结果才对。而对于非整数，例如10.5，就是错的。

错误的解法二:

```
IF Amt1 > 9 AND Amt1 < 14 THEN
    PRINT "It fits!"
END IF
```

显然，对于大于9和小于10（9<Amt1<10）的Amt1值，将错误地打印“It fits!”。这就是说，9.5不在10至13之间，但将打印“It fits!”。同样，若Amt1为13.5、或任何大于13且小于14的值，则错误地打印“It fits!”。因此，该IF块是错误的。

正确的解法：

```
IF Amt1 >= 10 AND Amt1 <= 13 THEN
    PRINT "It fits!"
END IF
```

这种解法不但正确而且简洁。

自我测验(四)

1～3中有无错误，若有请指出来。

1.
```
IF Amt = 5 OR 6 THEN
    PRINT "A-Ok"
END IF
```
2.
```
IF Amt1 OR Amt2 = 5 THEN
    PRINT "A-Ok"
END IF
```
3.
```
IF Amt1 = 5 OR IF Amt2 = 6 THEN
    PRINT "A-Ok"
END IF
```
4. 编写一个程序，若变量Amt1取值在[100，1000]内，打印"A-OK"。
5. 编写一个程序。当变量Amt1取值在(100，1000)内打印"A-OK"。

指出6～8题中的哪一个能在Total取值[5,10]内时，打印"A-OK"。

6.
```
IF Total > 5 AND Total < 6 THEN
    PRINT "A-Ok"
END IF
```
7.
```
IF Total > 4 AND Total < 11 THEN
    PRINT "A-Ok"
END IF
```
8.
```
IF Total >= 5 AND Total <= 10 THEN
    PRINT "A-Ok"
END IF
```

9. 某俱乐部只招收年龄大于等于21岁，且收入超过＄40000或者至少有＄5000存款的人员。用 IF-THEN-ELSE 语句表达该俱乐部的会员标准。

10. 用IF-THEN-ELSE语句表示如果你年终奖金多于＄2500，并且未付的账单少于＄2000，则显示"Go On Vacation"。否则显示"Stay Home This Year"。

对下面两题(11～12 题)，首先写出伪代码，然后编出 Turbo Basic 程序。

11. 输入每个雇员的性别，工资，打印(1)工资在＄50000和＄100000之间的女雇员的百分比。

(2) 工资在＄50000和＄100000之间的男雇员的百分比。首先写出伪代码，再编出 Turbo Basic 程序。

12. 输入某公司货名(Part Description)、单价(unit price)和库存数量(Quatity onhand) 打印货名和建议折扣率。

   (1) 单价在＄5～10之间，库存多于100的货物，折扣率为2%。
   (2) 单价在＄5～10之间，库存小于等于100的货物折扣率为1%。
   (3) 单价小于＄5，高于＄10，库存大于等于250的货物，折扣率为3%。
   (4) 其它的货物无折扣。

§4.3.4 取反条件句

§4.3.4.1 简单的取反条件句

所有的关系算式都能够取反：

```
IF  NOT  <条件>  THEN
       ⋮
```

带有 NOT 的关系运算符及其等效运算符如下表所述：

带有 NOT 的条件	等效的条件
IF NOT Amt < 7	IF Amt >= 7
IF NOT Amt > 7	IF Amt <= 7
IF NOT Amt <> 7	IF Amt = 7
IF NOT Amt = 7	IF Amt <> 7
IF NOT Amt >= 7	IF Amt < 7
IF NOT Amt <= 7	IF Amt > 7

下面的两条语句是等效的:

语句(a)

```
IF Amt >= 0 THEN
    PRINT "Amt is zero or positive"
ELSE
    PRINT "Amt is negative"
END IF
```

语句(b)

```
IF NOT Amt < 0 THEN
    PRINT "Amt is zero or positive"
ELSE
    PRINT "Amt is negative"
END IF
```

在任何情况下，NOT 必须紧随 IF 之后，然后才是被测试的条件。如果把:

```
IF NOT Amt = 7 THEN ...
```

写成:

```
IF Amt NOT = 7 THEN ...
```

将导致系统错误。

§ 4.3.4.2 取反复合条件句

使用取反的复合条件句时，需要特别小心!

例

编写一个程序，当输入的 Maritalcode$ 不是"S"(single)，或"D"(Divorced)时，打印"Married"；否则，打印"Unmarried"。为此，可编出如下的程序:

```
INPUT MaritialCode$
IF MaritialCode$ = "S" OR MaritialCode$ = "D" THEN
   PRINT "Unmarried"
ELSE
```

```
    PRINT "Married"
END IF
```

但是，假如要使用反条件句来编写，似乎简单地把每一个条件取反就能实现:

```
INPUT MaritialCode$
IF NOT MaritialCode$ = "S" OR NOT MaritialCode$ = "D" THEN
    PRINT "Unmarried"
ELSE
    PRINT "Married"
END IF
```

可实际情况并非如此。可以看出：取反后，下面的两个条件之一满足就可打印"Married"。

第一，MaritalCodes$≠"S"
第二，MaritalCode$≠"D"

假定 Maritalcode$ ="M"(Married)，计算机将显示"Married"，这是我们希望的结果。

然而，当 Maritalcode$ ="S"时，正确的结果应当显示"Unmarried"，但是该程序运行结果将显示"Married"。这是因为尽管此时条件 1 不满足，却满足条件 2。由于使用了"OR"，导致错误的结果。类似地，当"Marritalcode$ ="D"时，结果也是错误的。

为此，布尔代数中规定：由"OR"连接的条件取反时，除把每个条件取反外，还必须把连接词取反，即:"OR"改成"AND"。

```
IF NOT (<条件1> OR <条件2> OR ... )
```

等效为:

```
IF NOT <条件1> AND NOT <条件2> AND NOT ...
```

因此，上例正确的反条件形式程序为:

```
INPUT MaritialCode$
IF NOT MaritialCode$ = "S" AND NOT MaritialCode$ = "D" THEN
    PRINT "Unmarried"
ELSE
    PRINT "Married"
END IF
```

同样，由"AND"连接的条件取反时，除每个条件取反外，还应把"AND"改为"OR"。

复合条件取反时的替代方法

下述两语句是等效的:

```
1. IF NOT Amt1 = 3  AND  NOT Amt2 = 4  THEN
       PRINT "Ok"
   END IF

2. IF NOT (Amt1 = 3  OR  Amt2 = 4)  THEN
       PRINT "Ok"
   END IF
```

NOT 的布尔优先级高于 AND 和 OR，但低于括号. 在对复合条件取反时，需对 IF 后面的所有条件加括号后再取反.

例 1: 程序段:

```
IF NOT (Age > 12  AND  Age < 20)  THEN
    PRINT "Not a teenager"
END IF
```

例 2: 利用反条件句编写程序. 对于:

(1) 小于25岁，收于高于 $ 100000;

(2) 高于60岁，收入小于 $ 50000;

的雇员被认为是“例外”. 输入每个雇员的年龄、工资，并打印出例外的人数.

```
'This program prints Total Number of Exceptions
CLS
Total = 0
PRINT "Enter Age; Salary for each employee"
PRINT "Type 99, 999999 when Done"
INPUT Age, Salary
DO  WHILE Age < 99
    IF  NOT (Age >= 25) AND_
        NOT (Salary <= 100000) THEN
        Total = Total + 1
    END  IF
    IF  NOT (Age <= 60) AND_
        NOT (Salary >= 50000) THEN
        Total = Total + 1
    END IF
    INPUT Age, Salary
LOOP
PRINT "Total Number of Exceptions: "; Total
END
```

§ 4.4 利用行号和跟踪功能简化程序调试

如前所述，在 Turbo Basic 中，行号是可有可无的；尽管在其它版本的 Basic 中是必不可少的。使用行号的最大优点在于能使程序员跟踪正在执行的每条指令及其位置。请看下面的编码:

```
10    IF ...
20       :
30       :
40    ELSE
50       :
60       :
70    END IF
```

利用屏幕上的调试命令你可以置跟踪选择为 ON，参看图 4.1。

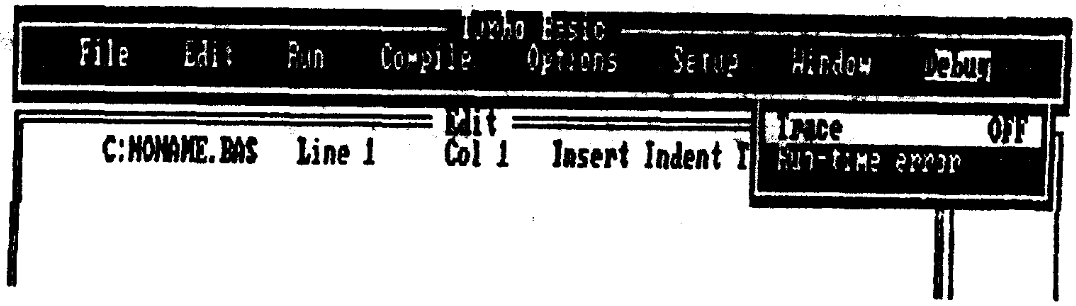


图 4.1 调试命令

如图 4.1 所示，Trace 选项开始时被置于 OFF，这是系统设定。当跟踪(Trace)选项为高亮度时，按回车键(Enter)即可显示 Trace On。

在 Trace 选项为 ON 的状态下，按 Esc 键，然后运行程序。在 Trace 窗口内，计算机将显示第一个执行的行号；按 ALT-F9 键，Trace 窗口将显示正在执行的每个行号。例如，Trace 窗内最先显示[10]，按 ALT-F9 键后显示[40][50]，这表明计算机在依次运行第 10、40、50 句；由此可知：IF 语句描述的条件不满足，因为第 20、30 句没有执行。

建议读者在最初调试程序时使用行号，并置 Trace 选项为 ON。这样便于调试，并可了解跟踪过程；也可以对含未查出错误的程序或其中的某一段标注行号，重新调试。

读者还可以把系统设置改为：Trace On，使计算机跟踪每一个有行号的程序。其方法是：首先选择 Trace On，然后选择 SetUp 命令。在此状态下，选择 Save / Options / Window / Setup，并按回车键，则计算机询问用户是否要改写 TBCONTIG.TB 文件；输入 Y，表示肯定。以后进入 Turbo Basic 程序时，只要不将

Trace 选项设为 OFF，那么它总是为: ON .

§ 4.5 本章辅导

§ 4.5.1 本章小结

一、简单关系符

A. 关系

$$\text{IF}\quad \text{操作数 1}\left\{\begin{matrix} < \\ > \\ = \\ \leqslant \\ \geqslant \\ \neq \end{matrix}\right\}\text{操作数 2}\quad \text{THEN}$$

操作数可以是变量，也可以是数值.

B. 如果IF块中的条件满足或为"真"，那么其后的语句将被执行，直至碰到END IF 或者 ELSE、ELSE IF 分句. 例如:

(a) 是否执行一个语句取于条件中是否为"真".

```
IF <条件> THEN
      语句组 1
ELSE
      语句组 2
END IF
```

(b) 是否执行数个语句，取决于是否条件为"真".

```
IF <条件> THEN
      语句 1
      语句 2
        ⋮
ELSE
      语句 3
      语句 4
        ⋮
END IF
```

在所有的IF语句块中，IF <条件> THEN 分句自己独占一行. ELSE 和

END IF 也分别独占一行。其它语句右移数列(Tab)以便于阅读和调制。

C. 如果IF块中的条件为"假"，将执行ELSE 或者ELSE IF 后面的语句(如果有)。如果没有 ELSE 和 ELSEIF 分句，则执行 END IF 后面的语句。

D. 数值比较

(a) 数字操作数的比较，采用代数规则：

12.0=12.00=+12

(b) 字符串操作数在进行逻辑比较时，应有相同的字符数。

(1) "ABC"与"ABC"只有二者长度相同时才相等。注意：

"b̸b̸ABC" < "b̸ABC" < "ABC"(b̸为空格)

(存储"b̸b̸ABC"，"b̸ABC"为一个字符串变量的数值，必须使用赋值语句或输入时带有引号。)

(2) 如果两个字符串操作数，直到较短的一个的最后一个字符均相同，则认为短串小于长串。即：

"ACB" < "ABCb̸" < "ABCb̸b̸" < "ABCD"

(c) 大多数微机使用的整理序列被称作ASCII码，根据ASCII码，数字小于字母，大写字母小于小写字母，即：

0 < 1 … < 9 < "A" … < "Z" < "a" … < "z"

二、其它类型的 IF 语句

A. 条件句嵌套：

```
(a)  IF  <条件1>  THEN
         IF  <条件2>  THEN
               ⋮
         ELSE
               ⋮
         END IF
            ⋮
     ELSE
            ⋮
     END  IF     (每个 IF 必须配有 END IF)
```

(b) IF <条件1> THEN

⋮

ELSE IF <条件 2> THEN

⋮

END IF (只有一个 END IF)

B. 复合条件句

(a) 格式:

IF <条件 1> OR <条件 2> THEN ...
IF <条件 2> AND <条件 2> THEN ...

(b) 优先级

(1) 如果OR 和AND用在同一语句中，首先从左至右计算AND，然后计算 OR.

(2) 括号优先级最高

C. 反条件句

(a) 任何判别都可以使用NOT，以判别条件的反. 如:

IF NOT Amt1 = Amt2 THEN ...

(b) IF NOT (Amt1 = Amt2) OR (Amt1 = Amt3)

与

IF NOT (Amt1 = Amt2) AND NOT (Amt1 = Amt3)

等价.

(c) NOT 的优先级高于OR和AND .

§4.5.2 本章自测

题 1～5 中有无错误? 若有，请纠正.

1.
```
IF A = B OR IF A = C THEN
    PRINT "A OK"
END IF
```

2.
```
IF Code$ = "Delete" THEN
    PRINT Ctr + 1
END IF
```

3.
```
IF C < A+B THEN
    PRINT "C OK"
END IF
```

4. IF A < 21 OR A = 21 AND A = 5 OR A > 5 THEN

```
        PRINT "A OK"
    END IF
```

5. IF A <> 3 OR A <> 4 THEN

```
        PRINT "A IS NOT 3 OR 4"
    END IF
```

6. 复合条件句的分级结构规定，用________连接的条件，首先计算，用________连接的条件其次计算，用________的优先级高于 AND 和 OR 。
7. 指出下面的两个语句是否等价?

```
(a)  IF  A < 3  OR  A > 4  THEN
          PRINT "Error"
     END  IF
(B)  IF  NOT  A = 3  OR  NOT  A = 4  THEN
          PRINT "Error"
     END  IF
```

8. 写出一个语句，当A取值于[3,13]，打印"A-OK"
9. 写出一个语句，当A取值于(3,13)时，打印"A-OK"
10. 写出一个语句，当下面的条件满足时打印"A-OK"，否则打印"NOT OK"

```
a  =  b
c  =  d
e  =-f
```

11. 编写一个程序，计算并打印A的绝对值.
12. 如果字符串变量Code $ 包括数值数据，打印"Is Numeric"；否则，打印"Is not Numeric"。
13 指出下段程序的运行结果:

```
IF  X=V+W/S-E+C+D  THEN
     PRINT  "Equal"
ELSE
     PRINT  "Unequal"
END IF
```

其中，X=30，V=30，W=10，S=5，E=20，C=3，D=6。
14. 写出一个IF块，当体重(weisht)大于120时，年龄(Age)大于等于25小于等于30(25 < age < 30)时，打印: "I am interested"。
15. 编写一个程序，计算每加伦汽油可行驶英里数. G为耗油量，T为行驶里程. 输

人这两个变量。如果 G 的输入值为 0，则不计算；若 G 的输入值是 99，就结束计算。要求先写出伪代码。

16. 编写一个程序，当A取值在[98.2, 100.6]时，打印"A-OK"，否则打印"A NOT OK"。
17. 下面的程序作为16题的解有无错误，为什么？

```
IF  A <= 100.6  THEN
    IF  A >= 98.2  THEN
         PRINT "A OK"
    END  IF
ELSE
    PRINT  "A NOT OK"
END IF
```

18. 写出一个程序找出A、B、C中最大的一个，并把结果存入PrintLarge。假定A, B, C 已经读入。首先写出伪代码。
19. 编写程序读入货款，如果贷款(Sales Amount) 大于 $ 500，折扣率5%；如果货款介于 $ 200-500 之间，折扣率为 3%；其它折扣率为 2%。打印每个货款的折扣。首先写出伪代码。
20. 编程计算周工资，首先输入每小时的报酬和工作时间，超过40小时的部分付50%的加班费。当输入的工作小时数是 99 时，结束计算。首先写出伪代码。

§ 4.5.3 实例

编写一个程序，输入下列变量:

```
PatientName $
SoreThroat( 1 = Yes, 0 = No)
LungInfection( 1 = Yes, 0 = No)
Sniffles( 1 = Yes, 0 = No)
Fever( 1 = Yes, 0 = No)
```

对于输入的每一组变量，用打印机打印出病人的姓名和诊断。图 4.2 给出了输入输出的样例。

注:

A. 输出是一个带有DIAGNOISIS REPORT 标题的打印报告。
B. 假定所有的病人至少有一种病症。
C. 如果一个病人同时有Lung Infection (肺炎)和Fever(发烧)两种病症，即可诊断为: PNEUMONIA。

D. 除C中的情况外，如果一个病人同时有两个或两个以上的病症，诊断是：COLD。

E. 如果病人只有一种病症，诊断是：OK。

其程序参见图 4.3。

输入数据

```
                      ┌──── Lung infection
                      │┌─── Fever
                      ││┌── Sniffles
                      │││┌─ Sore throat
                      ││││
F SINATRA             1111
J TRAVOLTA            1011
S STALLONE            1000
N STERN               0000
J CARSON              1100
A QUINN               1001
```

输出样例	诊断报告
F SINATRA	PNEUMONIA
J TRAVOLTA	COLD
S STALLONE	OK
N STERN	OK
J CARSON	PNEUMONIA
A QUINN	COLD
End of Reprot	

图 4.2 输入输出示范

```
CLS
LPRINT "                DIAGNOSIS REPORT"
LPRINT
INPUT PatientName$, LungInfection, Fever, Sniffles, SoreThroat
DO WHILE LungInfection <> 9
        IF LungInfection = 1 THEN
              IllFlag = 1
        END IF
```

```
            IF Fever = 1 THEN
                    IllFlag = IllFlag + 1
            END IF
            IF SoreThroat = 1 THEN
                    IllFlag = IllFlag + 1
            END IF
            IF Sniffles = 1 THEN
                    IllFlag = IllFlag + 1
            END IF
            IF IllFlag < 2 THEN
                    LPRINT PatientName$, "OK"
            ELSEIF LungInfection = 1 AND Fever = 1 THEN
                    LPRINT PatientName$, "PNEUMONIA"
            ELSE
                    LPRINT PatientName$, "COLD"
            END IF
            INPUT PatientName$, LungInfection, Fever, Sniffles, SoreThroat
            IllFlag = 0
LOOP
LPRINT "End of Report"
END
```

图 4.3　实例解答

§4.5.4　复习题

I　是非题

1. IF语句是用来实现迭行控制结构
2. IF语句中必须包括ELSE分句.
3. 在条件句中字符串变量不能和数值变量比较.
4. IF语句块中THEN必须和IF写在同一行.
5. 字符串变量比较中，其长度无关紧要.
6. 同一字母的大写和小写形式是相等的.
7. 在条件满足时，EXIT IF 可用于跳出IF语句块.
8. 在条件句嵌套中，如果有两个IF就必有两个END IF .
9. 在IF语句中同时有AND 和OR 则首先计算AND 连接的条件.
10. IF　NOT　(A　>　7)　THEN　…　　也可写成:
    IF　A　<　7　TNEN　…

II　综合题

指出 1～3 题中 Amt1 与 Amt2 的关系:

	Amt1	Amt2
1.	012	12
2.	12.0	12
3.	43	+43

4. 编写一个程序，计算FICA(特别安全税)．首先输入工资(Salary)，对于工资少于等于＄45000的税率为7.51%；若工资高于＄45000，则超过＄45000的部分不纳税．

5. 如果一个顾客购买少于12个面圈，单价为＄0.25，若多于或等于12个单价为＄0.18，写出一个程序，读入每个顾客的购买量并计算出应付金额．该程序应具有计算多个输入的能力。首先写出伪代码。

6. 某学校自助餐厅每杯咖啡售价＄0.6．如果每天售出量少于100杯，每杯成本为＄0.29；若多于或等于100杯，成本为＄0.24．编写一个程序，输入每天所售的杯数，并打印出每天的利润．要求首先写出伪代码．

§4.5.5 调试练习

在程序中使用题1和题2式的语句，指出是否有效?

1. IF Amt1 >= Amt THEN ... (注：用等于大于而不是大于等于)
2. IF Amt1 = Amt2 OR > Amt2 THEN ...
3. 置Trace选择为ON，指出下列程序段的结果．

a.
```
10  A1$ = "AABBCC"
20  Fred$ = "AABBCD"
30  IF  A1$ < Fred$ THEN
40    Barry$ = "5"
50  ELSE
60    Barry$ = "0"
70  END IF
```

b.
```
10  A$ = "Smith"
20  B$ = "Smith "
30  IF A$ < B$ THEN
40       PRINT A$; "<"; B$
50  ELSEIF A$ > B$ THEN
60       PRINT A$; ">"; B$
70  ELSE
80       PRINT A$; "="; B$
90  END IF
```

§4.5.6 编程题

1. 编写一个程序，首先输入变量值:顾客姓名、金额，对每一个顾客打印出包括下列各项的报告:

   CustomerName$(顾客姓名)
   SalesAmt(销售额)
   DiscountAmt(优惠额)
   NetAmount(应付额)

注:

(a) 如果销售额超过 $ 100,00，优惠率为3%。 如果销售额在 $ 50和 $ 100.00 之间，优惠率为 2%。

若销售额低于 $ 50.00，优惠率为 1%。

(b) 优惠额 = 销售额 * 优惠率

(c) 应付额 = 销售额 - 优惠额

2. 编写一个程序，计算并打印每个雇员的应得工资(GrossPay)，对每个雇员输入 EmpName $ (姓名)，HoursWorked (工作时间)和 Hourly Rate (每小时工资)三项，输出包括下述各项:

EmpName $ HoursWorked   HourlyRate   GrossPay

注:

(a) 应得工资 = 正常工作时间 * 每小时工资+加班时间 * 1.5 * 每小时工资

(b) 正常工作时数为40小时，超过部分为加班时间

3. 编写一个程序为模特儿介绍所打印两项总额:

a. 具有亚麻色头发，蓝眼，体重195-200磅，高于6英尺的男性总数。

b. 具有褐色头发，褐色眼睛，体重轻于120磅，身高5英尺2寸至5英尺6 寸的女性总数。

每个模特需输入以下各项:

模特编号:
头发颜色: (1 = 亚麻色，2 = 褐色，3 = 其它)
眼睛颜色: (1 = 亚麻色，2 = 褐色，3 = 其它)
身高: (英尺)
体重: (磅)
性别: (M = 男性，F = 女性)

4. 编写一个程序，为某大学打印一个学费报告，输入包括每个学生的studentName $ (姓名)和 NoOfcredits(学分)两项。如果一个学生选修小于或等于 12 个学分，每学分收取 $ 250.00 学费，若多于 12 个学分，学费均为 $ 3000.00，报告中应包括每个学生的学费和所有学生的学费总额。

5. 编一个程序，任意输入数据并打印总结报告。

a. 25 岁以下的司机所占的百分比。

b. 女性司机的百分比

c. 纽约司机的百分比

输入如下:

司机编码:
状态代码 (1～5):

出生日期 (MMYY).
性别 (M or F):

注:

状态代码 1 = NY

6. 某宠物商店出售猫、狗、鸟和热带鱼. 管理人员想知道哪种宠物最营利, 对每种商品输入下列内容:

宠物类型 (1 = 猫, 2 = 狗, 3 = 鸟, 4 = 鱼):
销售量:
商店实价:

报表应指明每种宠物类型:

PET	TOTAL NO. SOLD	TOTAL SALES	TOTAL PROFIT
猫			
狗			
鸟			
鱼			

每个商品的利润等于售价减去商店实价.

7. 编写一程序确定不同年龄用户的体育比赛票的花费. 用户的年龄作为输入, 如果年龄是 12 岁或者更小, 票的花费是 3 美元; 如果用户年龄大于 12 岁而小于 65 岁, 消费 6 美元; 如果用户是 65 岁或更大, 票的花费是 5 美元. 输入应是票的花费.

8. Olivia的父亲只允许她每月花15美元的长途电话费. Olivia想要一段程序算出她每一电话的花费. 下面是 Olivia 打的每种电话的代码和每分钟的花费:

代码	打 向	每分钟花费
A	Houston 她妹妹处	0.48
B	Rochester 她哥哥处	0.23
C	Pittsburgh 她的男朋友处	0.38

读入代码和电话的分钟数, 然后计算相应的花费。对每个电话, 应输出如下:

The Cost of this phone call is $ 3.43

9. Cherry计算机公司鼓励其职员学习大学课程, 提供给他们第一次800美元的学费的 70%, 第二次提供 500 美元的 50%, 第三次提供 300 美元的 30%, 写一段程序计

算输入的任一种学费的折扣数。

10. 两种体育杂志的价格给出如下:

	NewStand	Subscription
杂志 A	$ 2.50	$ 1.65
杂志 B	$ 2.95	$ 1.85

写一段程序，为购买每种杂志输入杂志代码（A或B）和地址代码（N-New stan 和 S = Subscription）。打印出关于每种杂志总售额的报表。

第五章　关于逻辑控制及自顶而下编程的迭代、循环和子程序

章要求:

1. 熟悉逻辑控制结构;
2. 掌握迭代和循环的使用方法;
3. 学会用GOSUB语句自顶而下模块化编程;

5.1　四种逻辑控制结构概述

不管什么编程语言，在所有程序中使用的逻辑控制结构如下:

逻辑控制结构	意　义
1. 顺序	程序按其出现的顺序执行.
2. 选择	IF-THEN-ELSE 选择一个或一些要执行的指令，取决于条件是否满足。
3. 迭代	反复执行一系列步骤；连续执行是基于给定的一个或一些条件是否满足。
4. 分情形选择	根据一个或一些变量的内容，执行许多步骤中的一步。

5.1.1　顺序结构

当 Turbo Basic 按指令顺序执行时，即不考虑任何给定条件的存在时，我们就称之为"顺序结构"。

顺序结构的伪代码是:

```
START
    :
    :    }    按语句出现的顺序执行
    :
STOP
```

Turbo Basic 程序为:

```
'This program prints an average grade
INPUT Exam1, Exam2, Exam3
LET Average = (Exam1 + Exam2 + Exam3) / 3
PRINT "The Average is "; Average
END
```

§5.1.2 选择结构

若要执行的 Turbo Basic 指令顺序依赖于某一特定的或一些条件时，我们称之为"选择结构"，并用 IF 语句块编程。

简单的 Turbo Basic 条件:

```
IF 条件 THEN
    :
    :
〔ELSE
    :
    :  〕
END IF
```

Basic 选择的伪代码与上面的 Turbo Basic 条件的格式相同，只是 ENDIF 是一个字。

简单条件的伪代码

```
IF 条件 THEN
    :
    :
〔ELSE
    :
    :  〕
```

```
ENDIF
```

在 Turbo Basic 中，嵌套条件具有下列格式:

```
IF  条件  THEN
    [IF  条件  THEN
        ⋮
        ELSE
        ⋮
        END IF]
[ELSE  IF  条件  THEN
        ⋮
        ⋮       ]
[ELSE
        ⋮
        ⋮     ]
END IF
```

嵌套条件的伪代码如下:

```
IF  条件  THEN
    IF  条件  THEN
        ⋮
    ELSE
        ⋮
    ENDIF
ELSE IF  条件  THEN
        ⋮
ELSE
        ⋮
ENDIF
```

§5.1.3 迭代结构

迭代，意即重复执行一系列步骤，直到满足给定条件为止。到现在为止，均使用 DO WHILE … LOOP 来执行迭代。作为复习，考虑下面的迭代：读入 Amt1 和 Amt2；如果 Amt1 不等于 9999(它是终值)，那么分别累加 Amt1 和 Amt2；这个过程迭代或重复执行，直到没有更多的数据；当 Amt1 输入且终值为 9999 时，满足条件。

```
LET Total1 = 0
```

```
LET Total2 = 0
DO WHILE Amt1 <> 9999
    LET Total1 = Total + Amt1
    LET Total2 = Total + Amt2
    INPUT Amt1, Amt2
LOOP
PRINT "Sum of Amt1 = "; Total1
PRINT "Sum of Amt2 = "; Total2
END
```

迭代的伪代码是.

```
DO   WHILE 仍有输入
  :
  :  }      按语句出现的顺序执行
  :
ENDDO
```

§5.1.4 分情形结构

当有许多后续路径需依据给定变量的内容而定时，应用分情形结构（case structure）。它是一种特殊的逻辑控制结构；例如，可以打印一个项目选单（Menu），用户用它选择要执行的操作。程序输入一个代表用户在选单上选项的变量，根据输入变量值的不同执行不同的功能。如果变量等于1，则打印总计；如果它等于2，则计算平均值，如此等等。根据下面变量的内容，用一个分情形结构选择所要求的过程：

```
SELECT CASE CodeIn
CASE = 1
    PRINT "Totals are "; Total1; Total2
CASE = 2
    LET Average = Total1 / Ctr
    PRINT "The Average Is "; Average
:
END SELECT
```

这种结构的伪代码如下：

```
SELECT   输入代码
   =1    打印总计
   =2    打印平均
     :
```

```
END  SELECT
```

如果用户理解了这些结构的每种构造，一旦学会了基本语法，就可以用任何语言进行编程，上面叙述的最后一种逻辑控制结构，即分情形结构将在第 8 章中详细讨论。

本章重点集中在迭代，将提供关于 DO WHILE … LOOP 的附加应用，并指出另外一些迭代方法。

§5.2 迭代方法

§5.2.1 关于迭代的 DO WHILE … LOOP 结构

§5.2.1.1 概述

DO WHILE（条件） … LOOP 反复执行，直到指定的条件不再是真或者满足。当循环终止时，继续执行 LOOP 后面的语句。

§5.2.1.2 如果在 DO WHILE … LOOP 中的条件测试开始时便不满足，那么该循环根本不执行。

注意，指定条件的测试是在一系列要执行的指令之前完成的，也就是如下考虑：

```
DO  WHILE Amt1<>9999
      ⋮
LOOP
```

假设 Amt1 最初输入 9999，意味着 Amt1 变量根本没用。DO WHILE … LOOP 内部的语句要执行之前，Amt1 与 9999 比较，由于 Amt1 开始时等于 9999，DO WHILE 的条件开始就不满足，在这种情况下，DO WHILE 内的语句根本不执行。那么将继续执行紧跟 LOOP 后面的语句。这是正确的过程；因为，如果一开始输入终值，那么就没有数据处理，就要越过整个循环。

§5.2.1.3 联合使用计数器与 DO WHILE，执行给定次数的循环

如上所述，重复或迭代执行一系列指令任意次数，就称作“循环”。用 DO WHILE Amt1<>9999，是由于事先不知道要处理的输入变量的个数。在这种情况下，可输入一个终值强制 DO WHILE … LOOP 终止。该终值在所有“实际”输入之后输入，而不管需要处理的输入变量个数有多少。

然而，有时用户事先精确知道要执行循环的次数。在这种情况下，可用一个计数器与 DO WHILE … LOOP 一起控制过程重复或迭代的次数，而不是靠测试终值来结束循环

——靠检查计数器来看是否完成了所要求的循环次数。

例 1:

若要精确输入 10 个数值变量并打印其总和，可以象通常那样“严格地”编程，即第十一次输入一个终值:

```
LET Total = 0
PRINT "Enter 10 Amounts - One per line"
PRINT "Then Enter 9999 on the 11th line"
INPUT Amount
DO  WHILE Amount <> 9999
    LET Total = Total + Amount
    INPUT Amount
LOOP
PRINT "The Sum of the 10 Amounts is "; Total
END
```

这个程序依赖用户在第十一次时输入 9999，因为 9999 是作为终值。如果用户不注意在第十二次或第十次输入 9999，计算机是不会发现这个错误的，象上面一样，文字“the Sum of the 10 Amounts is”将与累计值“Total”一起打印出来，也就是说，在前面的程序中，没有错误控制手段以保证恰好处理 10 个变量。

另外一种编程方式是提醒用户准确输入 10 个变量，并使计算机在第十个变量加上之后自行结束循环，这将免去使用终值并可准确输入 10 个变量。这是一种较好的编程方式，因为它检验输入的 10 个数值。

```
'Note that there is no initial read in this program
PRINT "Enter 10 Amounts--One Per Line"
LET Counter = 0
LET Total = 0
DO  WHILE Counter < 10
    INPUT Amount
    LET Total = Total + Amount
    LET Counter = Counter + 1
LOOP
PRINT "The Sum of the 10 Amounts is ; Total
END
```

在该程序中，使用了一个计数器跟踪 DO WHILE 迭代执行的次数。计数器初始化为 0，每通过一次循环加 1，因此，执行第一个循环之后，计数器为 1，第 2 次执行循环之后为 2，如此下去，当计数器为 10 时，10 个数值已经输入并累加完毕。DO WHILE 指定循环重复执行，直到不再小于 10，即它大于或等于 10。第十个变量加到总计上之后，计数器增至 10，循环终止。

注意，该程序用一个赋值语句将计数器（和总计）置为 0，这实际上并不必要，因为计算机将把所有的数值变量初始化为 0；但是，为了归档，最好将要累计（或增加）的变量初始化。

这个程序与以前的程序不同，没有在 DO WHILE 中测试终值，在循环之前也没有一个起始“INPUT”指令。这是因为，要读入和处理预先确定的固定输入个数，也就是

10. 若预先知道要处理的变量个数时，就没必要有一个终值或起始输入语句。

当要输入的变量数变化时，在本书提供的例子中还要使用（将继续使用）起始"INPUT"语句。例如，如果一个程序是要处理许多学生的考试成绩或清单上的大量零配件，则通常不能保证每次程序执行都有相同的输入个数；然而，如果预先知道要输入的变量个数，可以在终值的位置上使用一个计数器；象前面引用的程序一样，编写一个循环输入并处理每个变量。

例 2 使用计数器处理固定数目的输入数据。

假如输入用户数据如下：

用户姓名
街道地址
市、州、邮区

下面将为 25 个用户打印邮政标签：

```
PRINT "Enter Name on Line 1, Street Address on Line 2"
PRINT "Enter City State Zip on Line 3"
LET LabelCtr = 0
DO  WHILE LabelCtr < 25
     INPUT CustName$
     INPUT StAddr$
     INPUT CityStateZip$
     LPRINT CustName$
     LPRINT StAddr$
     LPRINT CityStateZip$
     LPRINT            'Prints a blank line between labels
     LET LabelCtr = LabelCtr + 1
```

下面是一个用于执行固定次数循环的迭代执行步骤清单：

固定循环次数的方法
1. 循环开始之前初始化计数器；
2. 每次执行循环，计数器增加；
3. 用"DO WHILE"语句测试计数器，保证循环执行要求的次数。

考虑下面程序段，看用户是否能发现它产生的逻辑错误：

问题： 总计 5 个输入的数值。

假定有下面的程序，它包含一个逻辑错误：

有逻辑错误的程序

```
LET Total = 0
LET Counter = 0
DO WHILE Counter < 5
    INPUT Amt1
    LET Total = Total + 1
LOOP
PRINT Total
END
```

程序中的 DO WHILE … LOOP 不对，因为在它内部的顺序执行步骤中没有指令增加计数器，因此，计数器初始化是 0，并一直保持为 0，因为计数器总小于 5，amt1 的值无限次地向总和上累加，这种错误情况叫做"无限循环"，也就是计算机将连续执行以上的过程，要求输入，直到程序员或用户认识到有错误并终止运行。

上述程序应正确编制如下：

```
LET Total = 0
LET Counter = 0
DO WHILE Counter < 5
    INPUT Amt1
    LET Total = Total + Amt1
     LET Counter = Counter + 1      ' Counter加 1
LOOP
PRINT Total
END
```

一旦计数器等于 5，DO WHILE … LOOP 将不再执行。打印出"总计"，并终止程序执行。

如果一个死循环的程序在主机上运行，计算机的内部时钟就会觉察到正在执行的语句长于任何程序的正常要求，将自动停止这项工作。在使用 Turbo Basic 的微型机上，可以按住"Ctrl"键，再按"Break"键退出死循环。但是，只有在第二章给出的 Options 命令中，选择了 Keyboard Break ON，这才行得通。在此，回顾一下这个选项。

为了在发生死循环时能终止处理，Keyboard Break 选项必须置为"ON"

如果在微型机上运行一个有死循环的 Turbo Basic 程序，当发生死循环时，程序员必须同时按下"Ctrl"键和"Break"键来中断处理。只有编译器配置成接受它们的这一目的，这些键才能停止程序的执行。正如第二章指出的那样，建议选用键盘中断"ON"(Keyboard Break ON)，以使"Ctrl-Break"能用来停止程序运行。这可以用下面三步过

程来完成.

---

1. 从主屏幕选择Options	Options 命令可通过先用"→"光标箭头加亮显示，然后按"Enter"键的方法来选择.
2. 从Options命令的下拉选单中选择: Keyboard Break ON	Keyboard Break OFF 是缺省的，是原始配置. 用"↓"把光标移到高亮度的 Keyboard Break OFF 处，把 Keyboard Break 置为"ON"，按"Enter"键. 然后按"ESC (ape) 键返回主屏幕.

3. 为了把这个选择永久性地存入Turbo Basic系统文件中，选择Setup命令并按"Enter"键，然后用"↓"光标选择: Save Options / Window / Setup 选项，并按"Enter"键.

4. 计算机将请用户认可:

Overwrite TBCONFIG.TB? (Y / N)

键入"Y"，将永久地保存Keyboard ON选择，使得按"Ctrl-Break"键总能终止程序的运行. 注意，"Break"键在大多数 IBM PC 机的 IBM 键盘上与"Scroll Lock"是同一个键；与大多数 IBM PS / 2 计算机的增强型键盘上的"Pause"键是同一个键.

---

§ 5.2.1.4 DO WHILE … LOOP 循环的附加实例

回忆一下，用户可以用 DO WHILE … LOOP 执行变次数或固定次数的循环.

例 1

为了进一步说明循环，让我们用一系列的连续加法，把两个数乘起来；当然，可以使用简单的乘（*）运算，但是为了说明循环方法，用 A 与自己相加 B 次来代替 A 乘 B. 例如，如果 A = 3，B = 4，则循环将 4 次把 3 加到一个总计上. 由于 3+3+3+3 = 3 × 4，结果将与使用乘法算子一样. 考虑下面的伪代码及程序:

伪代码

```
START
    读入两个变量 A 和 B
    初始化 Total
```

```
    DO  WHILE A > 0
        把 B 加到 Total 上
        A 减 1
    LOOP
    PRINT 结果
STOP
```

Turbo Basic 程序

```
INPUT A, B
LET Total = 0
DO WHILE A > 0
    Total = Total + 1
    A = A - 1
    PRINT NewSalary
LOOP
PRINT "The product of A×B is "; Total
END
```

假定 A=3，B=4，查看正确结果 12 是否存入 Total。

循环次数	各次循环后的结果		
	Total	**A**	**B**
1	4	2	4
2	8	1	4
3	12	0	4

一旦 A 不再大于 0，即它小于或等于 0 时，控制返回到 DO WHILE … LOOP 后面的语句，打印输出。

有必要强调指出，DO WHILE … LOOP 给定的条件在开始执行循环之前测试一次，然后在每次执行完循环后测试，这意味着，一开始条件就满足，那么循环根本不执行。那么，如果上面的 A 初始值为 0，则 DO WHILE … LOOP 将一次也不执行，并且"Total"保持它的 0 初始值。这就意味着前面的程序对于 A 的正值或 0，将正确工作，

如果 A 小于，它将不工作。

例 2

编写一个程序，用 DO WHILE … LOOP 计算并打印从 1 到 100 的整数之和，编制 Turbo Basic 程序并把结果与给定的比较。记住，程序可能稍有不同，但仍然是正确的，测试它以确定之。打印的结果应是 5050(参见图 5.1)。

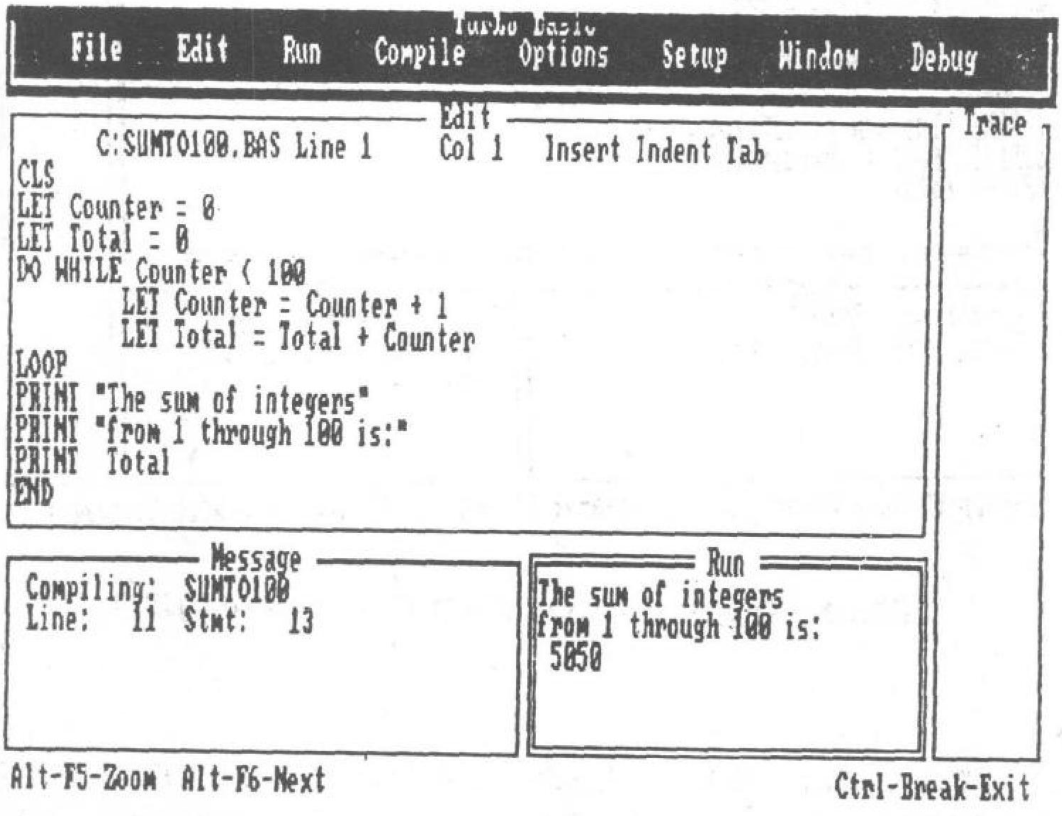


图 5.1 例题 2 使用 DO WHILE … LOOP 的程序

注意，该程序不要求提供输入，因为要处理的值已经知道（例如，1, 2, … 100)。

例 3

编写一段程序，用 DO WHILE … LOOP 计算并打印从 1 到 1001 的奇数之和。

写出用户自己的伪代码，然后用它编程。把结果与图 5.2 的程序比较。

这里也是如此，即没有必要给定输入值，因为这些值事先知道（如：1, 3, … 1001)。

在此说明中，首先把“Counter”加到“Total”上；这就意味着，在最后循环的尾部 Counter = 1001，还没有把 1001 加到“Total”上，所以，DO WHILE … LOOP 必须测试 Counter < = 1001，以加上最后一个值。

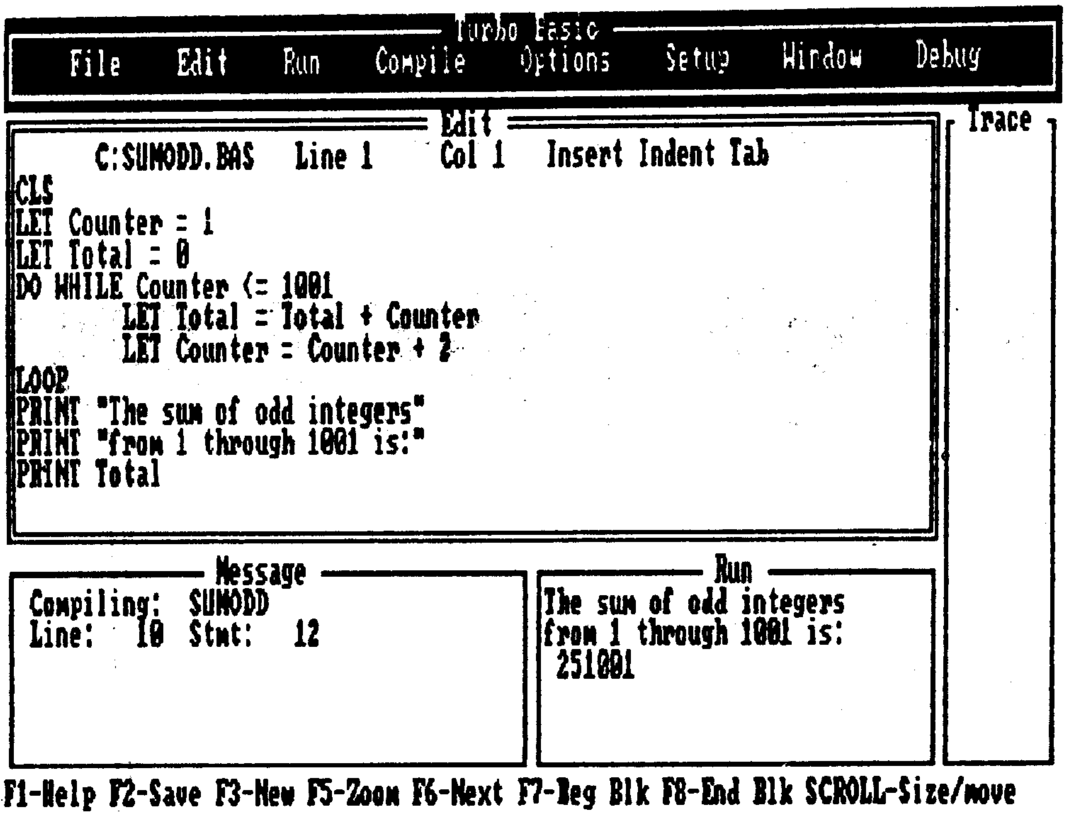


图 5.2 例题 3 使用 DO WHILE … LOOP 的程序

例 4

如果要输入的数值个数是一个变量时，通常使用一个起始输入语句，然后用 DO WHILE 测试终值．下面将提供一个程序样例，它不用起始输入语句和终值，处理输入值个数变化的情况．这个程序建立了一个称作 More Input $ 的变量，它开始时置为“Y”．先用 DO WHILE … LOOP 输入和处理输入值，然后问用户是否还有输入．如果有，用户键入“Y”，存入 MoreInput $；如果没有，用户键入“N”，它自动存入 MoreInput $，DO WHILE … LOOP 将反复执行，直到用户键入一个不是“Y”的值．

Edit

```
'This program inputs a variable number of amounts
'and prints a total
Total = 0
MoreInput$ = "Y"
DO  WHILE MoreInput$ = "Y"
    PRINT "Enter Amt:"
    INPUT Amt
    Total = Total + Amt
    PRINT "MoreInput (Y/N)?"
    INPUT MoreInput$
LOOP
PRINT "The Total is "; Total
END
```

Run

```
Enter Amt:
1500
More Input (Y/N)?
Y
Enter Amt:
1000
More Input (Y/N)?
N
The Total is 2500
```

用:

```
DO WHILE MoreInput$ = "Y"  OR MoreInput$ = "y"
```

可接受大、小写字母 y. 它可代替 DO WHILE 语句; 注意, 如果要用"否定"表示条件的话, 必须用"AND", 即:

```
DO WHILE MoreInput$ <> "N" AND MoreInput$ <> "n".
```

这个程序不需要起始输入语句, 因为在每个循环结束时都询问用户, 明确表示有还是没有其它输入.

自我测验(一)

在下面的各种情况 (1～5题) 中, DO WHILE … LOOP 各执行多少次?

1.
```
LET Ctr = 5
LET Total = 0
DO WHILE Ctr <= 10
    INPUT Amt
    LET Total = Total + Amt
    LET Ctr = Ctr + 1
LOOP
PRINT Total
```

2.
```
LET Ctr = 10
LET Total = 0
DO WHILE Ctr > 5
    INPUT Amt
    LET Total = Total + Amt
    LET Ctr = Ctr - 1
LOOP
```

```
PRINT Total
```

3.
```
LET Ctr = 10
LET Total = 0
DO WHILE Ctr < 10
    INPUT Amt
    LET Total = Total + Amt
    LET Ctr = Ctr + 1
LOOP
PRINT Total
```

4.
```
LET Ctr = 10
LET Total = 0
DO WHILE Ctr >= 10
    INPUT Amt
    LET Total = Total + Amt
LOOP
PRINT Total
```

5.
```
LET Ctr = 0
LET Total = 0
DO WHILE Ctr <= 5
    LET Ctr = Ctr + 1
    INPUT Amt
    LET Total = Total + Amt
LOOP
PRINT Total
```

6. **编写一段程序，读进个数可变的期终考试成绩，并打印一个班的平均值。不使用起始输入语句和终值，而是询问用户是否还有要处理的输入。**

7. **指出下面两个程序的不同点：**

a.
```
LET Total = 0
PRINT "Enter 25 Prices —— Then Enter 999"
INPUT Price
DO WHILE Price <> 999
    LET Total = Total + 1
    INPUT Price
LOOP
PRINT "Total Sales = "; Total
```

b.
```
LET Total = 0
LET Counter = 0
PRINT "Enter 25 Prices"
DO WHILE Counter < 25
    INPUT Price
    LET Total = Total + 1
    LET Counter= Counter + 1
LOOP
PRINT "Total Sales = "; Total
```

8. 写一段程序，计算并打印从2到2000的偶数之和。
9. (T或F) 如果DO WHILE … LOOP中指定的条件一开始不满足，那么发生死循环。
10. 考虑下面的程序:

```
Ctr= 0
DO WHILE Ctr <= 10
    :
    :
LOOP
```

(T或F)如果在DO WHILE … LOOP中，Ctr不改变，那么发生死循环。

§5.2.2 用于迭代的DO WHILE … LOOP结构变形

§5.2.2.1 WHILE … WEND

IBM 的 BASIC 和 BASICA 利用 WHILE … WEND 循环实现迭代或 DO WHILE … LOOP 结构。这种逻辑控制结构功能与 DO WHILE … LOOP 相象。Turbo Basic 允许用户使用上面描述的 DO WHILE … LOOP 或者 WHILE … WEND。

这些结构的格式非常相似:

DO WHILE … LOOP	WHILE … END
DO WHILE 表达式	WHILE 表达式
⋮	⋮
LOOP	WEND

例: 读入并累加5个数值。

```
'DO WHILE looping              'WHILE ... WEND looping
LET Total = 0                  LET Total = 0
I = 0                          I = 0
INPUT Amt                      INPUT Amt
DO  WHILE I < 5                WHILE I < 5
    Total = Total + Amt            Total = Total + Amt
    INPUT Amt                      INPUT Amt
    I = I + 1                      I = I + 1
LOOP                           WEND
END                            END
```

在这两种情况下，循环中I的值必须改变以便在某点不小于5。如果I的值不变化，程序将会死循环。

§5.2.2.2 DO … LOOP WHILE …

应注意，DO … LOOP 以及 WHILE … END 在执行循环之前，首先测试给定的条件，那么，在下面情况下，假设I=1。

```
DO  WHILE I > 1
       ⋮
LOOP
```

或:

```
WHILE I > 1
       ⋮
WEND
```

在这二种情况下，循环都不执行，因为I不大于1。

另一种方法可以确保 DO … WHILE 循环至少执行一次。为此，可编成 DO … WHILE 结构，使得执行完一次循环之后再测试条件。

```
DO                          DO 单独占一行
    ⋮  }
    ⋮  }    要执行的语句组
LOOP  WHILE I > 1
```

WHILE … WEND 不能以这种替代方式编程，因此 DO … WHILE 更灵活一些；它可以在循环的开始和结束处测试条件。

§5.2.2.3 DO UNTIL … LOOP 或 DO … LOOP UNTIL …

Turbo Basic 还允许用户用 DO UNTIL … LOOP；这种结构连续执行循环，直到给定条件满足为止；亦即：DO WHILE … LOOP 只要给定条件为“真”就执行迭代，而

DO UNTIL … LOOP 执行迭代直到给定条件为"真"．下面两种结构是等价的．

```
'DO WHILE looping
LET Total = 0
I = 0
INPUT Amt
DO  WHILE I < 5
    Total = Total + Amt
    INPUT Amt
    I = I + 1
LOOP
```

```
'DO UNTIL looping
LET Total = 0
I = 0
INPUT Amt
DO  UNTIL I >= 5
    Total = Total + Amt
    INPUT Amt
    I = I + 1
LOOP
```

在 I 小于 5 的时候执行循环与执行循环直到 I 大于等于 5 是一样的．这是完成同一功能的两种不同方式．

DO … UNTIL 与 DO … WHILE 类似，循环执行一次之后才测试条件：

```
DO                    ' DO 单独占一行
 :
LOOP UNTIL  I > 1
```

因此，DO WHILE … LOOP, DO UNTIL … LOOP 和 WHILE … WEND 都是实现迭代的逻辑控制结构，即反复执行一个指令序列．编程时，用户可以任意使用这些结构．但是，应注意，DO 循环更灵活一些，因为条件的测试可在循环执行之前或循环执行之后进行．

§ 5.2.3 在迭代中使用 EXIT 语句

有时有必要提前"退出"一个循环，也就是说，如果存在某种错误或其它例外的条件得到满足，可以在正常的终止之前退出循环．在此情况下可使用"EXIT"．如果打算从 DO WHILE, DO UNTIL 或 WHILE WEND 结构中退出，"EXIT"这个词后面必须跟"LOOP"这个词．

```
I = 0
DO  WHILE I < 5
    INPUT UnitPrice, Qty
    IF  UnitPrice < 0 THEN
        PRINT "Input Error - Run is Terminated"
        EXIT LOOP 'Will cause loop to terminate if UnitPrice < 0
    END IF
    LET Total = UnitPrice * Qty
    PRINT Total
    LET I = I + 1
LOOP
END
```

这个程序打印出 5 套输入数值的产品单价及数量．但是，如果在任意一点单价输入是

负数，就退出循环并停止程序执行.

因此，"EXIT LOOP"可用来提前退出 DO WHILE … LOOP，DO UNTIL … LOOP，或 WHILE … WEND. 实际上，EXIT"这个词可用来终止任意的逻辑控制结构. 就象第四章中所看到的那样，"EXIT IF"可以用在一个条件中，提前离开 IF … END IF"结构. 以后，用户将看到"EXIT"可用到其它结构中，例如：FOR … NEXT 以及 SELECT CASE 结构.

§5.2.4 嵌套迭代

所有的迭代形式均可嵌套，即：一个迭代可以出现在另一个迭代中.

例 1

假定要准备一些邮签. 每个用户需要 25 个，但是希望为大量的用户建立这样的邮签. 在这样的情况下，读入一行包括:

(1) 姓名；
(2) 街道地址；
(3) 市、州和邮区的输入数据.

假定城市、州和邮区之间没有逗号，对每个输入行打印 25 个邮签. 输入的用户姓名是"LAST"时，结束处理，即"LAST"是一个终值，表示输入数据输入结束. 在此，可以用一个起始输入语句和一个终值，因为不知道倒底有多少用户.

完成这一过程的一种方法是使用 DO WHILE（更多的数据）… LOOP，在它内部，再精确执行 25 次 DO WHILE. 在主 DO WHILE 内部，建立了一个计数器，其初始值为 0. 每写完一个邮签，计数器加 1. 这样，当计数器是 25 时，就知道已经为每组输入值打印了 25 个邮签. TURBO BASIC 程序如下:

```
'This program prints 25 mailing labels for each customer
CLS
INPUT CustName$, StAddr$, CityStateZip$
DO  WHILE CustName$ <> "LAST"
    LET Counter = 0
    DO  WHILE Counter < 25
        LPRINT CustName$
        LPRINT StAddr$
        LPRINT CityStateZip$
        LPRINT
        LET Counter = Counter + 1
    LOOP
    LPRINT 'Space after each set of labels
    LPRINT 'Space after each set of labels
    INPUT CustName$, StAddr$, CityStateZip$
LOOP    'Outer loop refers to 1st  DO WHILE
PRINT "End of Job"
END
```

这个程序说明了嵌套迭代的应用，主循环定义如下:

```
DO WHILE CustName$ <> "LAST"
  ⋮
LOOP
```

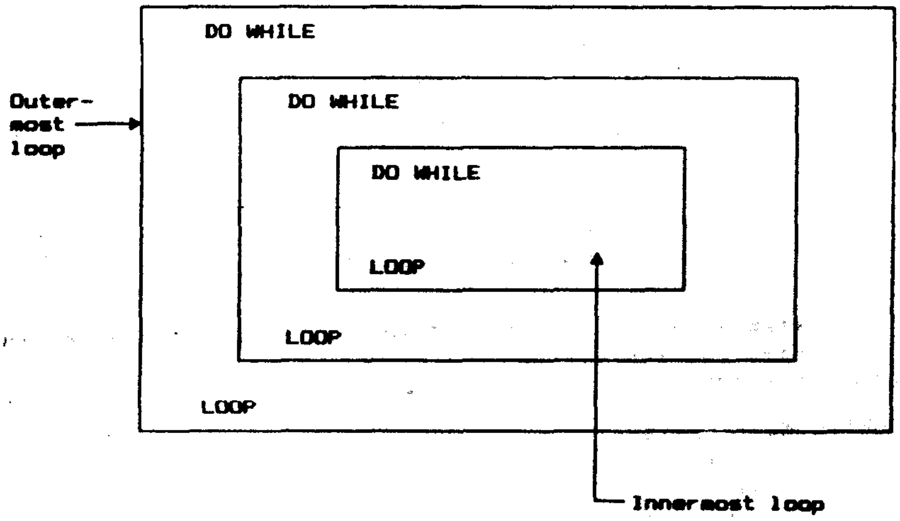
DO WHILE CustName$ < > "LAST" ··· LOOP 结构是主循环. 自开始处直到最后"LOOP"的所有指令都是它的内容. 只要 CustName 不等于"LAST", 整个结构将反复执行. 最后, CustName$ 等于"LAST"时, 程序流程转至 LOOP 语句的下面一条语句; 在此, 就是先打印"end-of-job"信息, 然后执行"END", 结束程序.

小的或内部用 DO WHILE Counter < 25 ··· LOOP 定义. 在小循环或模块执行 25 次之前, "Counter"必须置为 0, 然后执行小循环或模块 25 次, 或者直到 Counter = 25. 在打完第一个邮签之后, "Counter"置为 1; 第 2 个邮签打完后, 置为 2, 如此下去. 这就是说, 写完 25 个邮签之后, "Counter"将是 25, 小的或内部 DO WHILE ··· LOOP 就不再执行了, 程序流程转向第一个或内部"LOOP"之后的下一指令. 紧跟的两个"LPRINT"在两个邮签之间给出两个空行. 然后读入下一组变量值. 整个过程反复执行, 直到终值 CustName$ 等于输入值: "LAST", 程序才结束.

注意, 上面的 DO WHILE CustName$ < > "LAST"也可以编写成 DO UNTIL CustName$ = "LAST", 而 DO WHILE Counter< 25 可编成 DO UNTIL Counter= 25.

尽管没必要将"Counter"初始化为 0, 因为所有数值变量将自动置 0, 但是, 外层循环要求这样做. 这是因为, 不仅在开始时要求 Counter 置 0, 而且每次通过主循环时也要求"Counter"置 0.

注意, 也可以把"Counter"初始化为 1 而不是 0. 程序写成 DO WHILE Counter< =25.



上面所述的在 DO WHILE 里面用 DO WHILE, 称做"嵌套 DO WHILE". 主或

外层 DO　WHILE 通常意味着，只要有输入就执行整个过程或模块．这个过程或模块自己可能还有某些迭代或指令序列要反复执行，称作另外的 DO　WHILE．记住：最后的或外层“LOOP”对应着第一个“DO　WHILE”，而内部的“LOOP”对应着内部的“DO　WHILE”．建议用户将指令明确地写入相应的循环：

编程时，应记住每个“DO”必须与最后的“LOOP”匹配．

例 2

假定有 10 个考试成绩输入，每个学生对应一行，想打印每个学生的平均成绩，但不知道班上有多少学生．在此，将用 DO　UNTIL … LOOP 来说明程序的结构．

由于不知道班里有多少学生，所以主 DO　UNTIL … LOOP 的执行次数是一个变量，小循环将使用一个计数器，因为对每个学生要处理 10 个考试成绩．

我们知道将在小循环内处理每个输入行的 10 个数据．读入并处理完 10 个输入成绩之后，将询问用户是否还要输入．主循环是否继续处理取决于用户的回答，伪代码如下：

伪代码

```
START
    将 MoreInput$ 置为“Y”
    DO  UNTIL  MoreInput$  <>  “Y”
        初始化"Couter"为"Total"
        DO  UNTIL Counter=10
            键入考试成绩
            将成绩加至“Total”
        计数器 Counter 加 1
        ENDDO
        计算并打印“平均”
        给 MorInput$ 键入“Y”或“N”
    ENDDO
STOP
```

Turbo Basic 程序

```
'This program prints an average of 10 exam grades for each student
CLS   'Clears the run window
MoreInput$ = "Y"

DO  UNTIL MoreInput$ <> "Y"
        Counter = 0
        Total = 0
        DO  UNTIL Counter = 10
                INPUT Exam
                Total = Total + Exam
                Counter = Counter + 1
        LOOP
```

```
            PRINT "The Average is "; Total / 10
            PRINT "More Input (Y/N)?"
            INPUT MoreInput$
    LOOP
    PRINT "End of Job"
    END
```

注意：MoreInput $ 是一个变量或“标志”，初始化为“Y”，输入 10 个考试成绩之后，将提示用户：“MoreInput （Y／N）？”，如果用户回答大写字母“Y”，则继续处理下面的 10 个成绩，直到用户回答不是大写字母“Y”。

上面的计数器计数处理的考试成绩，从 0 到 10 变化；可以通过附加的学生计数器打印下面的平均成绩，使得读起来更加方便：

```
The Average for Student [1] is xx.x
The Average for Student [2] is xx.x
```

下面程序建立的学生计数器称作：“StudentCounter”，成绩计数器称作“ExamCounter”：

```
'This program prints an average of 10 exam grades for each student
CLS    'Clears the run window
MoreInput$ = "Y"
StudCounter = 0    'Varies from 1 to n students
DO  UNTIL MoreInput$ <> "Y"
    ExamCounter = 0    'Varies from 1 to 10 and is then reset
    Total = 0
    DO  UNTIL ExamCounter = 10
        INPUT Exam
        Total = Total + Exam
        ExamCounter = ExamCounter + 1
    LOOP
    StudCounter = StudCounter + 1
    PRINT "The Average for Student "; StudCounter; " is "; Total / 10
    PRINT "More Input (Y/N)?"
    INPUT MoreInput$
LOOP
PRINT "End of Job"
END
```

用这两个计数器，也可以打印一个班的平均成绩：

```
'**********************************************************************
'This program prints an average of 10 exam grades for each student
'It also prints a class average
'**********************************************************************
CLS    'Clears the run window
MoreInput$ = "Y"
```

```
StudCounter = 0    'Varies from 1 to n students
ClassTotal = 0
DO  UNTIL MoreInput$ <> "Y"
    ExamCounter = 0    'Varies from 1 to 10 and is then reset
    Total = 0
    DO  UNTIL ExamCounter = 10
        INPUT Exam
        Total = Total + Exam
        ExamCounter = ExamCounter + 1
    LOOP
    StudCounter = StudCounter + 1
    PRINT "The Average for Student "; StudCounter; " is "; Total / 10
    ClassTotal = ClassTotal + Total / 10
    PRINT "More Input (Y/N)?"
    INPUT MoreInput$
LOOP
PRINT "Class Average is "; ClassTotal / StudCounter
PRINT "End of Job"
END
```

然后，计算机就执行如下的嵌套迭代:

Number of Times Through Main Loop	StudCounter	ExamCounter
1 ⋮ 10	1 ⋮ 1	1 ⋮ 10
11 ⋮ 20	2 ⋮ 2	1 ⋮ 10
21 ⋮ 30	3 ⋮ 3	1 ⋮ 10
⋮	⋮	⋮

自我测验(二)

1. 在上面的例2中，为了继续处理，必须给MoreInput $ 输入一个大写字母“Y”. 重编主 DO … WHILE，使得用小写字母“y”，也能使处理继续.
2. 由于逻辑错误，循环反复执行时，这叫做________循环.
3. 一个DO WHILE循环可用________语句来终止.
4. (T或F) DO WHILE和DO UNTIL可用来完成相同的功能.
5. DO WHILE用来实现称作________的控制结构.
6. ________语句可用来在循环正常结束之前，终止循环的处理.
7. (T或F) 所有迭代方式均可嵌套.
8. 到目前为止讨论的迭代形式有________，________和________.
9. 编制下面的程序，先用DO WHILE编，再用DO UNTIL重编一次.
   造一个表格，给出一周20个小时工作量的总费用，酬金从每小时4美元起，并以0·25 美元的增量增加，直到每小时 6 美元，表格的形式如下：

```
Wage Report for 20 Hours Per Week
Hourly Pay     Weekly Pay
 4               80
 4.25            85
 4.5             90
 4.75            95
 5              100
 5.25           105
 5.5            110
 5.75           115
 6              120
End of Report
```

10. 用WHILE …… WEND编制下面的程序，累加从1到99的所有奇数并打印结果.

§5.3 FOR … NEXT 语句

§5.3.1 FOR … NEXT 作为另一种迭代方法

总之，循环是一种反复执行的一个指令序列固定次数的一种迭代方式，前面已经用过关于迭代的好几种循环方式，在本节中，将讨论 FOR … NEXT 循环.

如果以给定的利率存了一定量的钱，现在假定希望计算出 1、2、3、年底累计钱的数量，假定利息是每年复合的. 意思是在第 2 年里,新的本金是起始本金加上第一年的利息，因而返回的利率将为该增加过的本金为基础.

C-220 BM

公式如下：

1 年后的本金 = 起始本金× (1+利率)
2 年后的本金 = 超始本金× $(1+利率)^2$
3 年后的本金 = 超始本金× $(1+利率)^3$

计算复合利息的一般公式为：

$$P_N = P_0 (1+r)^N$$

其中：P_0 = 超始本金 ·
P_N = N 年后的本金.
r = 利率
N = 年数

当 N 为 1 时，P_N 等于在给定利率 (r) 下的最初投资 (P_0) 一年后的总金额；当 N 为 2 时，P_N 等于投资两年后的总量；N=3 时，P_N 等于投资三年后的总金额.

下面以 3 年期限的复合利息为重点，来说明问题.
输入包括本金 (P_0) 和利率 (r)；要投资的年数预定义为3.
输出将包括一个图表，指定3年投资中每年的本金加上利息. 下面给出了一个显示输入和输出的例子.

```
ENTER PRINCIPAL AMOUNT:
? [2000]
ENTER INTEREST RATE: (e.g., 8% AS 8.0):
? [10.0]

INVESTMENT CHART FOR 2000 AT 10 % INTEREST
     YEAR      AMOUNT
      1         2200
      2         2420
      3         2662
```

现在，编写一个计算总值的程序，其中 P (本金) 和 R (利率) 作为输入值键入，要求计算 N 等于 1，2，3 时的结果.

方法 1：不用循环

```
CLS
PRINT "ENTER PRINCIPAL AMOUNT:"
INPUT Principal
PRINT "ENTER INTEREST RATE (e.g., 8% AS 8.0):"
INPUT Rate
PRINT
PRINT "INVESTMENT CHART FOR"; Principal; "AT"; Rate; "% INTEREST"
PRINT "       YEAR      AMOUNT"
LET Rate = Rate / 100
```

```
LET Amount1 = Principal * (1 + Rate) ^ 1
PRINT "        1        "; Amount1
LET Amount2 = Principal * (1 + Rate) ^ 2
PRINT "        2        "; Amount2
LET Amount3 = Principal * (1 + Rate) ^ 3
PRINT "        3        "; Amount3
END
```

在第六章，用户将看到如何用美元和美分数字表示所有的值。

可以看到，方框内的输入有很多相似之处。算术运算使用了同一基本公式。对于第 1 年，(1+Rate) 乘以本金，第二年用 $(1+Rate)^2$ 或 (1+Rate) × (1+Rate) 乘以本金，第三年用 $(1+Rate)^3$，或 (1+Rate) × (1+Rate) × (1+Rate) 乘以本金。

对于一个由增量集重复的指令序列最好用反复执行单一指令序列的一个循环来编程。

上面的问题可以编写一个过程计算三个值。这个过程使用了一个叫做“No OFYears”的变量指标，例如，它从 1 开始，每执行一次过程，增加 1，直到变量超过 3，也就是建立一个循环并执行三次，从 1 到 3 变化“NoOfyears”。

方法 2: 使用 DO WHILE … LOOP

```
CLS
PRINT "ENTER PRINCIPAL AMOUNT:"
INPUT Principal
PRINT "ENTER INTEREST RATE (e.g., 8% AS 8.0):"
INPUT Rate
PRINT
PRINT "INVESTMENT CHART FOR"; Principal; "AT"; Rate; "% INTEREST"
PRINT "      YEAR    !  AMOUNT"
LET Rate = Rate / 100
LET NoOfYears = 1
DO WHILE NoOfYears <= 3
        LET Amount = Principal * (1 + Rate) ^ NoOfYears
        PRINT "      "; NoOfYears; "      "; Amount
        LET NoOfYears = NoOfYears + 1
LOOP
END
```

可以看到，该程序使用了 DO … WHILE 循环，要重复执行的指令在 DO WHILE 过程体内。因为“NoOfYears”起始值不大于 3，所以 DO WHILE 内的语句第一次执行时，NoOfYears=1，然后把 1 加到 NoOfYears 上，再次计算测试条件，看 NoOfYears 是否大于 3。因为此时它是 2，不大于 3，所以 DO WHILE 内的语句在 NoOfYears=2 时再执行一次；然后 NoOfYears=3，DO WHILE 循环再执行一次。经过第三次循环之后，1 加至 NoOfYears，使它等于 4；由于大于 3，所以循环不再执行并且程序运行结束。

也可以用 DO UNTIL … LOOP 或 WHILE … WEND 完成同一类型的循环。

方法3: 使用FOR … NEXT循环

从上述方法中可以看出，要执行一系列的操作，可使用一个NoOfYears；它从1开始计数，直到等于3．要执行一个预定次数循环时，通常使用FOR … NEXT语句．在这种情况下，要求有一个变量进行计数．

FOR … NEXT结构可执行好几种功能:

(1) 建立一个变量;
(2) 增加该变量;
(3) 指定一个变量的变化范围;

那么，对于上述问题，有如下程序:

```
FOR NoOfYears= 1 TO 3
   LET Amount = Principal * (1 + Rate) ^ NoOfYears
   PRINT "       "; NoOfYears; "      "; Amount
NEXT NoOfYears
END
```

变量NoOfYears=1，第一次执行FOR … NEXT内的语句．指令FOR NoOfYears= 1To 3表示首先在NoOfyears=2，然后等于3时，重复执行这些语句．

因此,该过程执行3次，首先NoOfyears=1，然后NoOfYears= 2，最后NoOfYears=3．然后程序继续执行"NEXT"后面的语句，在这里是程序的结束．FOR … NEXT循环没必要把计数器(NoOfYears)开始置为1，并且每执行一次循环增加1，这是由FOR … NEXT结构自行处理的．

使用FOR … NEXT循环的全部程序如下:

```
CLS
PRINT "ENTER PRINCIPAL AMOUNT:"
INPUT Principal
PRINT "ENTER INTEREST RATE (e.g., 8% AS 8.0):"
INPUT Rate
PRINT
PRINT "INVESTMENT CHART FOR"; Principal; "AT"; Rate; "% INTEREST"
PRINT "      YEAR     AMOUNT"
LET Rate = Rate / 100
FOR NoOfYears = 1 TO 3
      LET Amount = Principal * (1 + Rate) ^ NoOfYears
      PRINT "       "; NoOfYears; "      "; Amount
NEXT NoOfYears
END
```

不管使用哪种方法编程，输入输出显示都是一样的．考虑到输出值不能打印成美元和美分数字，第六章将把重点放到编程打印输出上，以打印出两位十进制数．

不管重复执行指令序列的次数是固定的，或者是一个每执行一次都增值的变量，都可使用FOR … NEXT．

§ 5.3.2 FOR … NEXT 的详细说明

§ 5.3.2.1 变量定界符

到目前为止，在 FOR … NEXT 循环中变量 NoOfYears 的起点和终点使用的都是数字（例如， FOR NoOfYears = 1 TO 3）. FOR … NEXT 的起点和终点、或定界符也可以是数值变量；而且，起始定界符要求不必为 1.

例:

```
1. LET S = 1
   FOR N = S TO 10        'N will vary from 1 to 10
       .
       .
       .
   NEXT N
2. LET T = 5
   FOR N = 1 TO T         'N will vary from 1 to 5
       .
       .
       .
   NEXT N
3. LET U = 5
   LET V = 15
   FOR N = U TO V         'N will vary from 5 to 15
       .
       .
       .
   NEXT N
4. I = 6
   FOR N = 1 TO I-1
       .
       .
       .
   NEXT N 'N WILL VARY FROM 1 TO 5
```

重新考虑利率问题，其中储蓄的年数作为输入，并用作变量定界符.

Edit

```
PRINT "ENTER PRINCIPAL AMOUNT:"
INPUT Principal
PRINT "ENTER INTEREST RATE (e.g., 8% AS 8.0):"
INPUT Rate
PRINT "ENTER NO. OF YEARS OF INVESTMENT:"
INPUT NoOfYears
PRINT
PRINT "INVESTMENT CHART FOR"; Principal; "AT"; Rate; "% INTEREST"
PRINT "        YEAR        AMOUNT"
LET Rate = Rate / 100
    FOR N = 1 TO NoOfYears
        LET Amount = Principal * (1 + Rate) ^ N
        PRINT "       "; N; "    "; Amount
    NEXT N
END
```

```
Run
ENTER PRINCIPAL AMOUNT:
? 450
ENTER INTEREST RATE (e.g., 8% as 8.0):
? 9.0
ENTER NO. OF YEARS OF INVESTMENT:
? 10
```

RUN 窗口显示完上述信息后，接着给出如下信息:

```
INVESTMENT CHART FOR 450 AT 9% INTEREST
     YEAR      AMOUNT
      1         490.5
      2         534.644
      3         582.761
      4         635.209
      5         692.377
      6         754.691
      7         822.612
      8         896.648
      9         977.345
      10         1065.31
```

根据用户所使用的 Turbo Basic 版本的不同，输出值可能舍入不同。

顺便提示一下，下一章将集中介绍把数值显示成美元、美分形式及其在标题下定位的方法。

§5.3.2.2 如果 FOR 循环的起始定界符大于结束定界符，则不执行循环

例1: 假定要打印邮签，但是希望允许用户决定要求的邮签数量。为此，可编程如下:

```
PRINT "Enter No. of Mailing Labels Desired: "
INPUT NoOfLabels
INPUT CustName$, Address1$, Address2$
FOR N = 1 TO NoOfLabels
    LPRINT CustName$
    LPRINT Address1$
    LPRINT Address2$
    LPRINT
    LPRINT
NEXT N
END
```

如果起始定界符大于结束定界符，那么 FOR … NEXT 循环内的指令将被整个跳

过；即：如果用户对输入提示键入 0，那么 NoOfLabels 等于 0，语句 FOR N = 1 TO 0 将导致结果为 0 或不执行循环.

例 2. **FOR … NEXT 内的变量**

先读入表示输入变量个数的数值，然后读取每个变量的值，用FOR … NEXT打印出最大的变量值.

```
PRINT "Enter number of Amounts"
INPUT N
PRINT "Enter Amounts"
INPUT Amt1
LET Hold = Amt1
FOR I = 1 TO N-1
    INPUT Amt1
    IF  Hold < Amt1 THEN
        Hold = Amt1
    END IF
NEXT I
PRINT "Largest number is "; Hold
```

第一个 Amt 存入 Hold，每个附加值键入之后，与 Hold 比较，如果它大于 Hold，则它替换 Hold，否则，不作处理. N 个数值读进之后，Hold 将包括最大值.

数据样例

```
N = 5
Amt1 = 3
Amt1 = 8
Amt1 = 2
Amt1 = 9
Amt1 = 6
```

1. Amt1存入Hold (Hold = 3)
2. 循环再执行4次，输入处理其它4个量.
3. 第二个Amt1大于Hold (8 > 3)，所以，第2个Amt1替换Hold，Hold 现在是 8.
4. 第三个Amt1小于Hold，所以不处理它.
5. 第4个Amt1大于Hold，所以它替换Hold，现在Hold是9.
6. 第5个Amt1小于Hold，所以不处理它.

处理完所有输入之后，Hold 包含的是 9；即，5 个值中的最大值.

§ 5.3.2.3 在 FOR … NEXT 循环中使用不是 1 的增量

对于：

```
FOR I = 1 TO 5
```

```
    ⋮
NEXT  I
```

计算机执行循环 5 次，第一次 I 的值为 1，然后是 2，直至达到 5. 当 I=5 时，先第 5 次执行循环，然后执行 NEXT 后面的语句.

因此，每执行一次循环，I 都增加 1. 使用"STEP"分句，I 的增量可以是所希望的任何值.

假定要计算 1 到 101 之间所有奇数的和；因终点已知，故采用 FOR … NEXT 循环. 编程如下:

```
LET T = 0
FOR I = 1 TO 101 STEP 2
    LET T = T + I
NEXT I
PRINT "The Sum of Odd Numbers from 1-101 is "; T
```

在 FOR … NEXT 循环中，I 等于 1，T 的增量为 I，而 I 的增量是 2. 下面给出操作过程:

循环次数	执行循环后的 I 值	T
0	1	0
1	3	1
2	5	1 + 3
⋮	⋮	⋮
50	103	1 + 3 + 5 + … 101

STEP 分句可以指定为一个变量而不是一个值，那么前面的例子可编程如下:

```
LET Q = 2
FOR I = 1 TO 101 STEP -2
    LET T = T + 1
NEXT I
```

进一步，可以用 STEP 分句减少该变量的值；下面的程序也是正确的.

```
FOR I = 101 TO 1 STEP -2
    LET T = T + 1
NEXT I
```

如果 STEP 分句省略，缺省 STEP 值是 1，即计算机默认每执行一次该变量加 1，所以下面的两个指令会产生相同的结果.

```
FOR I = 1 TO 5          FOR I = 1 TO 5 STEP 1
```

要使I从5开始减到1，那么就必须使用STEP—1:

```
FOR I = 5 TO 1 STEP -1
```

FOR … NEXT中的变量不要求是整数，下面情况是允许的.

```
FOR I = .5 TO 2.5 STEP .5
```

但是应注意：用整数变量作为计数器变量，FOR … NEXT循环的执行速度要快一些. 如果在FOR … NEXT中使用的变量假定仅为整数值，可以用一个"%"作为该变量名的一部分，来特别定义该变量:

```
FOR I% = 1 TO 5
```

I%仅是整数变量，建议用户在FOR … NEXT循环中使用整数变量.

不要在FOR … NEXT自己的循环内改变它的变量值，下面的程序可能会产生错误:

```
FOR I = 1 TO 5
    .
    .
    .
    '*****************************************
    '* I should not be used as a receiving *
    '* variable in a LET statement          *
    '*****************************************
    LET I = ...
NEXT I
```

即：I用作FOR … NEXT的循环计数器，程序员不应该在上面的赋值语句中将它用作接收变量.

注意，上面的注释用"*"框起来，以突出注释. 象前文中一样，继续使用这种方法以突出注释部分.

§5.3.3.4　不要在循环外使用由FOR … NEXT定义的变量

考虑如下程序:

```
FOR I = 1 TO 5
        :
NEXT I
PRINT I      ' 循环之外不应访问变量I
```

由于FOR … NEXT中的I从1变到5，用户可能会认为上面程序执行时打印5或者6，事实上，I仅在FOR … NEXT内部具有可预见的值，在循环外不应访问它.

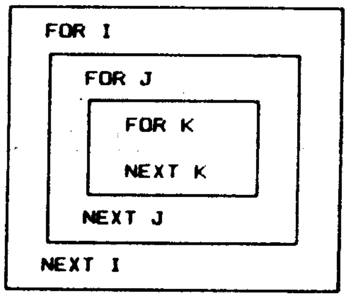
§ 5.3.3.5 嵌套的 FOR … NEXT

FOR … NEXT 循环与所有其它迭代结构一样，可以嵌套，意思是说在 FOR … NEXT 内包括 FOR … NEXT，当使用嵌套的 FOR … NEXT 循环时，且记每个 NEXT 与最近的 FOR 匹配. 考虑下面的结构:

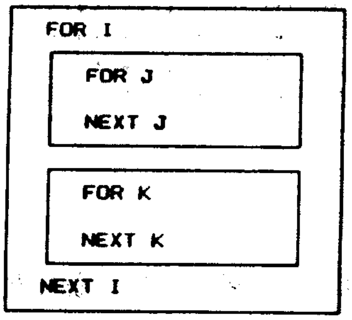
有效结构

（图中代码的缩进仅仅是为了结构的明了）

a.



b.



无效结构

a.

```
FOR I
  FOR J
NEXT I
  NEXT J
```

b.

```
FOR I
  FOR J
    FOR K
NEXT I
  NEXT J
    NEXT K
```

还应注意，下面的程序将产生语法错误:

```
FOR  N  =  1  to  5
     ⋮
FOR  M  =  1  to  10
     ⋮
NEXT  N
     ⋮
NEXT  M
```

NEXT N 与 NEXT M 指令必须对调，才能使这个结构正常运行。即：起始的 FOR N= 1 TO 5 是主循环，必须与最外层的 NEXT N 链接。小的或经二个 FOR M=1 TO 10 应与第一个或内部的 NEXT M 键接。如果小 FOR … NEXT 循环写入到前面的循环，则会很容易检查到错误。

嵌套 FOR … NEXT 的应用举例

读入 20 组变量，各组包括：

姓名
街道地址
市州邮区

对每组变量打印 5 个邮签。

现在用 FOR … NEXT 嵌套编写这一程序。为每组数据打印 5 个邮签应是小循环，读取 20 组变量应是主循环。

开始用自顶而下方法编程，首先结构化主循环，然后填入细节。

伪代码

第一步

```
FOR MajorLoopCtr  =  1  TO  20
    输入一组变量
    打印 5 个邮签
NEXT  MajorLoopCtr
```

现在，主结构已确定，可以填入细节。

第二步

```
FOR  MajorLoopCtr = 1  TO  20
```

```
输入一组变量
FOR MinorLoopCtr = 1 TO 5
    打印一个邮签
NEXT MinorLoopCtr
NEXT MajorLoopCtr
```

上面已设计出描述程序逻辑的伪代码，下面便可以用 Turbo Basic 开始编程:

Turbo Basic 程序

```
FOR MajorLoopCtr = 1 TO 20
    INPUT Name1$, Address1$, Address2$
    FOR MinorLoopCtr = 1 TO 5
        LPRINT Name1$
        LPRINT Address1$
        LPRINT Address2$
        LPRINT                    'Two blank lines print
        LPRINT                    'after each label
    NEXT MinorLoopCtr
NEXT MajorLoopCtr
END
```

该程序用嵌套 FOR … NEXT 循环读进 20 行输入，对每行输入产生 5 个邮签. FOR … NEXT 迭代结构是最适用于准确知道要处理的输入行数这种情况.

回顾自顶而下编程的概念和伪代码

可注意到，用自顶而下的方法设计程序结构，意即从主结构开始，然后一步步填入细节. 这种方法的另外一个术语叫做"逐步求精法". 此外，用户还可将伪代码用作描绘逻辑结构的规划工具. 根据伪代码，应用循环结构设计程序. 但在规划阶段，没有必要将重点集中在 Turbo Basic 的基本语法规则上. 伪代码使开发程序时，在没有真正编程之前，没必要关心语法.

编程原则

可注意到：作为编程原则，必须把一个 FOR … NEXT 缩进到另一个 FOR … NEXT 中，而且在小循环中缩进所有语句，使得程序更易于阅读和调试. 按一次 Tab 键，在一行内缩进 8 个空格；第二次按 Tab 键，缩进另外 8 个空格. 或者，换用空格键也可缩进空格.

通读本文，用户将更加清楚下述几种方法的优点:

(1) 用伪代码作为规划工具;
(2) 自顶而下方法作为设计方法;
(3) 为易于阅读和调试，将逻辑控制结构中的语句缩进。

§5.3.3.6 EXIT与FOR … NEXT连用

象其它所有逻辑控制结构一样，FOR … NEXT循环也可包含EXIT语句来提前退出循环。这通常在某个错误条件发生的情况下使用。

例:

累加25个数值，并打印总和。不过，如果有某数值等于0，则打印当时的总计，并终止程序的运行。

```
INPUT Amt
FOR I = 1 TO 25
    IF  Amt = 0 THEN
        EXIT FOR     'If Amt is 0 the loop is terminated
    END IF
    Total = Total + Amt
    INPUT Amt
NEXT I
PRINT Total
END
```

除非Amt等于0，否则FOR … NEXT执行25次。Amt为0时，程序流程转向NEXT后面的语句。

§5.3.3.7 FOR … NEXT的完整格式

下面是关于FOR … NEXT语句的格式:

```
FOR 变量 = X TO Y [STEP Z]
    ⋮      语句组
NEXT  [变量[, 变量] … ]
```

若干个变量包含在NEXT语句中意思是: 下面两段程序节选产生相同的结果:

优选格式	可接受的等效格式
FOR N = 1 TO 10	FOR N = 1 TO 10
FOR M = 1 TO 20	FOR M = 1 TO 20
⋮	⋮
NEXT M	NEXT M, N

NEXT N

但是，应注意到，对于 NEXT M，N，嵌套中 M 必须在 N 的前面。

§5.4 迭代方法的选择

一种迭代结构可用 DO WHILE，DO WHILE … WEND，DO UNTIL 或者 FOR … NEXT 来编程。

当迭代次数是未知的或是一个变量时，大多数程序员更愿意用 DO WHILE … LOOP, DO UNTIL … LOOP 或 WHILE … WEND 循环。即：如果迭代次数依赖输入，最好用 DO. . . WHILE 方案，或者是（DO … WHILE，DO … UNTIL，或者是 WHILE … WEND）。但是，如果执行迭代的次数是固定的，最好使用 FOR … NEXT。

对于 FOR … NEXT，变量或索引的值自动改变，而在任何 DO … WHILE 方案中，程序员必须增加（或减少）这些变量或索引的值。

还应记住，即使条件测试一开始就满足（或不满足），仍要至少执行一次循环，则用 DO … LOOP UNTIL … ，或 DO … LOOP WHILE …

自我测验(三)

1. 下面程序有什么错误?

```
FOR N = 1 TO 4
        INPUT Amt
        PRINT Amt
NEXT
```

2. 用FOR … NEXT循环，打印一个如下所述的倍增表:

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

3. 用DO WHILE … LOOP给上题编程.
4. 下述FOR … NEXT循环嵌套正确吗?

```
FOR X1 = …
     ⋮
```

```
        FOR  X2 =  ···
              ⋮
        NEXT  X2
        FOR  X3 =  ···
              ⋮
        NEXT  X3
    NEXT  X1
```

5. 用DO UNTIL编程下面的问题

```
CLS
Total = 0
Ctr = 1
DO WHILE Ctr <= 10
    INPUT Amt
    Total = Total + Amt
    Ctr = Ctr + 1
LOOP
PRINT Total
END
```

6. 用FOR … NEXT编程上面的问题.
7. 使用FOR … NEXT，编写一程序计算并打印人2到2000（包括端点）的所有的偶数之和.
8. 编写程序输入两个整数变量N%和M%。然后计算并打印人N%到M%的整数之和. 例如：如果 N%=4 且 M%=7，则打印 4，9（4+5），15（4+5+6），22（4+5+6+7）.
9. 计算N的阶乘，其中N是一个输入变量，N的阶乘 = N×（N−1）× … 1（例如 5!=5× ×4×3×2×1=120）.
10. 考虑下面的程序:

```
INPUT X, Y, Z
FOR I = X TO Y/Z STEP Z*X
        PRINT I
NEXT I
END
```

如果输入如下，那么程序的打印结果是什么?

a. 1，10，1
b. 5，5，1

c. 10, 1, -. 5
d. 2, 12, 1
e. 1, 2, 3

§ 5.5 在子程序中引入自顶而下模块化编程方法

§ 5.5.1 为什么使用子程序

到目前为止，本书集中讨论了 Turbo Basic 的语法和程序设计所必须的四种逻辑控制结构，还学习了如何在开始编程之前用伪代码规划程序逻辑，而且引入了自顶而下编程的概念。

本节将详细讨论"自顶而下"、"模块化"程序设计方法。自顶而下模块化程序易于编写、调试、维护和修改。将一个模块定义为一个过程或一组起着单元功用的指令，执行一组特殊的操作。在自顶而下的 Turbo Basic 程序中引进模块的一种方法是使用子程序(subroutines)。

子程序有以下特征：

§ 5.5.1.1 子程序可以减少重复性工作

考虑下面的程序段：

```
PRINT "Enter Customer No:"
INPUT CustNo
PRINT "Enter Part Description:"
INPUT PartDescription$
PRINT "Enter Unit Price:"
INPUT UnitPrice
PRINT "Enter Qty Ordered:"
INPUT QtyOrdered
DO  WHILE CustNo < 999
    LET BalDue = UnitPrice * QtyOrdered
    LPRINT "Customer "; CustNo; "Owes: "
    LPRINT BalDue; "For "; "Part "; PartDescription$
    PRINT "Enter Customer No:"
    INPUT CustNo
    PRINT "Enter Part Description:"
    INPUT PartDescription$
    PRINT "Enter Unit Price:"
    INPUT UnitPrice
    PRINT "Enter Qty Ordered:"
    INPUT QtyOrdered
LOOP
LPRINT "End of Report"
END
```

每个方框内的内容表示一个完成给定的功能的过程或指令序列，在这里，过程给出输入提示及键入输入值.

可注意到两个方框的内容完全一样，它们出现在两个地方——作为起始读入过程的一部分，然后，又出现在 DO WHILE … LOOP 的尾部.

重复方框内的指令集两次，不仅麻烦，而且增加了出错的风险. 子程序可以写成一个独立的语句序列或模块，然后，在程序中任何需要的地方执行.

GOSUB 语句引起一个独立过程的执行，然后将程序流程返回到 GOSUB 后面语句. GOSUB Label 指令执行由 Label 指定的子程序中的语句，因而可以写成:

```
GOSUB PromptAndGetInput
DO  WHILE PartNo < 999
    LET BalDue = UnitPrice * QtyOrdered
    LPRINT "Customer "; CustNo; "Owes: "
    LPRINT BalDue; "For "; "Part "; PartDescription$
GOSUB PromptAndGetInput
LOOP
LPRINT "End of Report"
END
```

GOSUB PromptAndGetInput 指令指出计算机执行一个模块. 该模块本身将跟在 END 语句后面. GOSUB 引起执行模块中的所有指令. PromptAndGetInput 模块内的所有语句执行完之后，程序流程返回到主程序中 GOSUB 后面的语句，在上面的程序中是 DO WHILE 语句.

```
GOSUB  PromptAndGetInput:
   :
END

PromptAndGetInput:
   :
   :  }  将要执行的指令系列
RETURN    ←  程序流程返回到 GOSUB 后面的语句
```

用标号定义的子程序编写在 END 语句之后，并且必须用 RETURN 作为结束语句. 在这里，标号 PromptAndGetInput 后跟一个冒号，并且写在一行上.

RETURN 也写成独立的一行.

```
GOSUB PromptAndGetInput
DO   WHILE CustNo < 999
     LET BalDue = UnitPrice * QtyOrdered
     LPRINT "Customer "; CustNo; "Owes: "
     LPRINT BalDue; " For "; "Part "; PartDescription$
     GOSUB PromptAndGetInput
LOOP
LPRINT "End of Report"
END
PromptAndGetInput:
     PRINT "Enter Customer No:"
     INPUT CustNo
     PRINT "Enter Part Description:"
     INPUT PartDescription$
     PRINT "Enter Unit Price:"
     INPUT UnitPrice
     PRINT "Enter Qty Ordered:"
     INPUT QtyOrdered
RETURN
```

可注意到子程序 PromptAndGetInput 在主程序中执行了两次，然而，子程序的实际指令序列仅在程序的尾部编写了一次．END 语句后面可以包括任意多个子程序．直到现在为止，END 语句一直是任选的．但是，如果使用子程序，就要求有 END 语句，以分离主程序和这些子程序．

总之，在程序中出现一次以上的指令序列可以编成独立的子程序．通过主程序中的 GOSUB 指令执行之．如果一个指令序列表现为一紧密结构．那么它即使出现一次，也最好编成子程序．

例:

```
IF Amt1 = Amt2 THEN
    :
    :  }   Amt1 等于 Amt2 时要执行的指令序列.
    :
ELSE
    :
    :  }   Amt1 与 Amt2 不相等时要执行的指令序列.
    :
END IF
```

每个序列都是一个紧密结合的指令集，将在一定条件下统一执行．因此，上面程序的结构的改进形式如下:

```
IF Amt1 = Amt2 THEN
        GOSUB    EqualAmtsProcessing
ELSE
        GOSUB    UnequalAmtsProcessing
```

END IF

§ 5.5.1.2 子程序用于自顶而下模块化程序设计

在规划如上所述的程序时，开始就集中考虑带下划线指令的语法是不明智的，最好是首先集中考虑程序的总体结构，开发程序的逻辑．伪代码是用于程序规划的工具，它强调程序的逻辑，而不是它的语法。

考虑下面的语法格式:

伪代码

```
START
    PERFORM
        打印提示及相应的输入数据
    ENDPERFORM
DO  WHIL 还有输入
        计算 BalDue
        打印结果
        PERFORM
            打印提示及相应的输入数据
        ENDPERFORM
    ENDDO
STOP
```

可以看到以上伪代码集中考虑程序的总体结构，不是具体细节．在伪代码中，PERFORM … ENDPERFORM 结构表示要在程序中编程的独立模块或子程序，完成指定的任务。

一旦用伪代码建立了程序的总体结构，程序员就可开始编程了．但是，编程同规划一样，最好是集中考虑总体结构，而将细节放在后面考虑．在上面的情况中，伪代码中指定的 PERFORM … ENDPERFORM 的地方最好使用 GOSUB.

因此，当编制一个程序时，在主程序中用 GOSUB 标明希望后面详细编程的子程序或指令序列，这种方法叫做自顶而下程序．这里每个子程序叫做一个模块．长而复杂的程序可能包含以下主程序，它将所有细节留给子程序．

```
GOSUB InitializeVariables
GOSUB PromptAndGetInput
GOSUB PrintHeadings
DO  WHILE Amt < 999
    GOSUB ProcessInput
    GOSUB PrintOutput
```

```
    GOSUB PromptAndGetInput
LOOP
GOSUB PrintTotals
END

InitializeVariables:
    :     { Separate module goes here
RETURN
PromptAndGetInput:
    :     { Separate module goes here
RETURN
PrintHeadings:
    :     { Separate module goes here
RETURN
ProcessInput:
    :     { Separate module goes here
RETURN
PrintOutput:
    :     { Separate module goes here
RETURN
PrintTotals:
    :     { Separate module goes here
RETURN
```

建议用户采用这种方式编制自顶而下程序，即：任何表现为相关操作集合的指令序称，如打印标题或输入一系列变量，均应编制成模块或子程序．这种自顶而下方法的目的就是使程序员集中考虑程序结构，而不是程序主体中的语法，把细节留给子程序．类似地，在程序中多处执行的指令序列也应编成子程序，并通过主程序中的 GOSUB 语句执行之．

其它几种模块化方法也可用于减少重复性工作和达到自顶而下的目的，这些叫做"过程"和"函数"；第十一章和第十二章将深入讨论这些问题．本节将集中讨论作为一种自顶而下方法的 GOSUB，因为这是最简单的．

§ 5.5.2　GOSUB 的格式

GOSUB 的格式如下：

```
    :
GOSUB  Label
    :
END                    主程序结束
```

```
Label :              ' 标号后面为冒号
   ⋮
RETURN
```

GOSUB 语句引起执行从命名标号处开始的一个指令序列，当所有从标号处开始，到 RETURN 处结束的指令执行完以后，程序流程转向直接跟随 GOSUB 的语句。

标号包含有字母、数字和作为字分隔符的句点。标号中间不允许有空格，切记不要使用 Turbo Basic 的保留字，如 Error , EXIT 和 Reset 作为标号. 关于 TurboBasic 保留字的清单见附录。

例:

```
'This program prints amounts - 25 per page
CLS
GOSUB PrintHeadings
INPUT Amt1
DO  WHILE Amt1 < 999
    LPRINT "Amount = "; Amt1
    LineCtr = LineCtr + 1
    'LineCtr keeps track of the number of lines printed
    IF  LineCtr = 25 THEN
        GOSUB PrintHeadings   'We want 25 lines per page
    END IF
    INPUT Amt1
LOOP
END
PrintHeadings:
    NewPage$ = CHR$(12)
    LPRINT NewPage$
    'In the next chapter we see that the above two
    'instructions advance the paper to a new page
    LPRINT "       DETAIL REPORT"
    LineCtr = 0          'Reinitializes Line Counter
RETURN
```

PrintHeadings 在程序的开始处执行；然后，当 LineCtr= 25 时再次执行。PrintHeadings 执行到 RETURN 之后，程序流程返回到 GOSUB 后面的语句。起初，执行完 PrintHeadings 之后，程序流程返回到第一个 INPUT 语句。执行完循环内的 GOSUB PrintHeadings 之后，程序流程返回到循环内的 INPUT Amt1 FS 语句。

另例

考虑下面的子程序

```
AddTwo:
    SumOfOddNumbers = SumOfOddNumbers + OddNum
    OddNum = OddNum + 2
RETURN
```

1. 用上面的子程序，计算从1到1001的所有的奇数之和，包括两个端点.

```
SumOfOddNumbers = 0
OddNum = 1
DO WHILE OddNum <= 1001
    GOSUB  AddTwo
LOOP
PRINT "Sum of Odd Numbers from 1～1001 is "; SumOfOddNumbers
END

AddTwo:
      SumOfOddNumbers = SumOfOddNumbers + OddNum
      OddNum = OddNum + 2
RETURN
```

2. 用上面的子程序，计算从11到2001的所有奇数之和，包括两个端点.

```
SumOfOddNumbers = 0
OddNum = 11
DO WHILE OddNum <= 2001
    GOSUB  AddTwo
LOOP
PRINT "Sum of Odd Numbers from 1～1001 is "; SumOfOddNumbers
END

AddTwo:
      SumOfOddNumbers = SumOfOddNumbers + OddNum
      OddNum = OddNum + 2
RETURN
```

在这两种情况下，Add Two … RETURN 都在 END 语句的后面编写. 用这种方法，不管是数值或是字符串数据均可传向和来自子程序.

§5.5.3 嵌套 GOSUB

所有逻辑控制结构均可嵌套，意思是任何这种结构均可在其内部再包含这种结构. 例如: 在一个由 GOSUB 调用执行的模块内可再次使用 GOSUB 语句:

```
 :
GOSUB  SUB1
```

```
  ⋮
END

SUB1:
    ⋮
    GOSUB Sub2
    ⋮
RETURN

SUB2:
    ⋮
    GOSUB SUB3
    ⋮
RETURN

SUB3:
  ⋮
RETURN
```

加下划线的 GOSUB 叫做嵌套 GOSUB，因为它们包含在由 GOSUB 调用执行的模块。

嵌套 GOSUB 的例子

```
:
GOSUB PrintHeadings
:

DO  WHILE Amt1 < 999
    IF  Amt1 < 0 THEN
        GOSUB ErrorRoutine
    ELSE
        GOSUB ProcessData
    END IF
    INPUT Amt1
LOOP
GOSUB FinalTotalRoutine
END
PrintHeadings:
    NewPage$ = CHR$(12)    'These two instructions skip
    LPRINT NewPage$        'to a new page--see Chapter 6
    LPRINT "          DETAIL REPORT"
    LPRINT "         FOR COMPANY.ABC"
    LineCtr = 0
RETURN
```

```
ErrorRoutine:
    PRINT "Amount is Negative"
    PRINT "Re-Enter Input"
RETURN
ProcessData:
    .
    .
    .
    LPRINT Amt1; Amt2; Amt3
    LineCtr = LineCtr + 1
    IF  LineCtr = 25 THEN
        GOSUB PrintHeadings      'This is a nested GOSUB
    ENDIF
    .
    .
    .
RETURN
FinalTotalRoutine:
    .
    .
    .
    LPRINT FinTot
    LineCtr = LinCtr + 1
    IF  LineCtr = 25 THEN
        GOSUB PrintHeadings      This is a nested GOSUB
    ENDIF
    .
    .
    .
RETURN
```

§5.5.4 程序中 GOSUB 的放置

在程序中，END 语句后面的子程序或模块的编写顺序将不影响程序的编译或执行。但是，建议首先编写要调用其它模块的模块；类似地，我们也极力主张：主模块在小模块之前编写；这样，主模块首先在程序中出现。尽管如此，并不强求这么做。

例:

本例可以正常运行，但是建议不这样做:

```
    ⋮
GOSUB  Sub1
    ⋮
END

Sub2:                   ' 小模块
    ⋮
RETURN

Sub1:                   ' 主模块
```

```
    ⋮                    ' 它还调用 Sub1
    GOSUB  Sub2
    ⋮
RETURN
```

由于 SUb1 调用 Sub2，所以建议在程序中首先编写 Sub1。

优选格式

```
    ⋮
GOSUB  Sub1
    ⋮
END

Sub1:
    ⋮
    GOSUB  Sub2
    ⋮
RETURN

Sub2:
    ⋮
RETURN
```

因此，自顶而下程序应以降序包含模块，从主模块到小模块，要调用其它模块的模块首先编写。

```
GOSUB  Initialize
    ⋮
GOSUB  InputVariables
DO  WHILE  Amt  <  999
      GOSUB  ProcessVariables
      GOSUB  PrintResults
      GOSUB  InputVariables
LOOP
GOSUB  PrintTotals
END

Initialize:
    ⋮
```

```
    GOSUB  PrintHeadings

    ⋮
RETURN

InputVariables:
    ⋮
RETURN

ProcessVariables:
    ⋮
RETURN

PrintResults:
    ⋮
    IF  LineCtr  =  25  THEN
        GOSUB  PrintHeadings

    END  IF
    ⋮
RETURN

PrintTotal:
    ⋮
    IF  LineCtr  =  25  THEN
        GOSUB  PrintHeadings

    END  IF
    ⋮
RETURN

PrintHeadings:
    ⋮
RETURN
```

尽管允许用行号代替或附加到标号上，但仍要用标号标识 GOSUB。

建议用户使用带有描述性标号的 GOSUB … RETURN 结构，这些描述性标号已在本章中说明，是用来标识 GOSUB 的。

```
      ⋮
GOSUB  PrintRoutine
      ⋮
END

PrintRoutine:          ' 这是一个描述性的标题
      ⋮ }              ' 子程序中的语句均应缩进
      ⋮
RETURN
```

如果希望跟踪程序的逻辑，也可以将行号与标号一起使用.

```
         ⋮
 50  GOSUB  PrintRoutine
 60       ⋮
100  END

110  PrintRoutine:         ' 这是一个描述性的标题
          ⋮ }              ' 子程序中的语句均应缩进
          ⋮
200  RETURN
```

如果在上面的程序中，使用了 Debug 命令的 Trace ON 选项，则跟踪窗口打印如下:

```
. . .  [50]  [PrintRoutine]
. . .  [200]  [60]   . . .
```

也可用行号代替标号标识 GOSUB 的入口点，但建议不要这样编程. 观察以下程序，它用的是"GOSUB　行号"形式，而不是"GOSUB　标号"形式:

正确——但不推荐使用

```
  10  GOSUB  1000
         ⋮
  50  END
1000  PRINT  "Hello!"
1010  RETURN
```

象以前指出的那样，很多 Basic 版本要求有行号. 而 Turbo Basic 允许有行号，但不

要求有；除非程序逻辑控制结构复杂，用户希望跟踪，或者发生逻辑错误时而不能确定其位置，否则建议用户不要使用行号．在这种情况下，用户可以添加上行号，并使用 Debug 命令的 Trace ON 选项，这样，实际执行的行号将显示出来，这是一种逐步执行程序而发现错误的非常有用的方法．然而，在其它所有情况下都省略了行号，总是使用描述性标号标识 GOSUB 的入口点．

在一个具有嵌套 GOSUB 的程序中，跟踪程序逻辑结构举例。

用嵌套 GOSUB 执行的指令顺序如下(其中用行号标识带标号的 GOSUB，以便跟踪程序逻辑)：

```
 10
 20
 30 GOSUB  Sub1
 40 :
    :
100 END
110 Sub1:
120 ...
130     GOSUB  Sub2
140 ...
150 RETURN
160 Sub2:
170 ...
180 RETURN
```

如果 Debug 命令有 Trace ON 选项，则跟踪窗口将显示如下：

```
------------- Trace -------------
[10] [20] [30] [SUB1] [120] [130]
[SUB2] [170] [180] [140] [150]
[40]
```

回顾一下，置 Trace 选项"ON"的方法为：

1. 高亮度突出Debug命令，并按回车键：

   File Edit Run compile Options Setup Window <u>Debug</u>

2. 加亮Trace选项：

   如果 Trace OFF 高亮度显示，则按回车键，选用 Trace ON：

Trace	ON

3. 按 Esc 键退出Debug命令

§ 5.5.5 常见的 GOSUB 错误

§ 5.5.5.1 GOSUB 没有包含 RETURN 将导致逻辑错误

RETURN 是使用 GOSUB 语句时所要求的定界行．如果没有它，程序将连续执行，直到找到一个 RETURN．如果程序中只有一个 GOSUB 语句，忽略 RETURN 语句将会引起语法错误．但如果有很多 GOSUB 语句，那么有可能不发生语法错误；然而，程序的执行是不正确的．

逻辑错误举例: 没有 RETURN 的子程序

```
GOSUB InputAA
DO  WHILE Amt < 999
    GOSUB PrintBB
    GOSUB InputAA
LOOP
END
InputAA:
    INPUT Amt           'RETURN should follow the INPUT
PrintBB:
    LPRINT "Amt = "; Amt
RETURN
```

可以看到，InputAA 子程序没有 RETURN 语句，该程序不发生语法错误．但是，执行结果如下:

Run

```
? 10
10
? 20
```

Printed Output

```
Amt = 10
Amt = 10
Amt = 20
Amt =
  .
  .
  .
```

第一次执行 InputAA 子程序，计算机连续处理，直到碰到一个 RETURN；这意味着 InputAA 和 PrintBB 都被执行过．从 InputAA 到 PrintBB 的所有语句执行完之后，控制返回到主程序中的 DO WHILE 内，此时，DO WHILE 内的第一条指令 GOSUB PrintBB 又一次被执行，因此，对每个输入值，PintBB 执行二次．

总之，应切记在每个子程序尾部要有 RETURN 语句，以避免发生上述逻辑错

误.

§ 5.5.5.2　在该有 END 语句的地方没用它将会发生错误

应该注意，END 语句在程序中第一个标号（与 GOSUB 对应）之前，如果不这样将导致逻辑错误.

常见错误举例

1. 程序中使用了GOSUB语句，但没有END 语句.

```
GOSUB  PrintHello
INPUT  Amt
DO  WHILE  Amt  <  999
      LPRINT  Amt
      INPUT    Amt
LOOP
PrintHello:
    LPRINT  "Hello!"
RETURN
```

最初执行GOSUB 时，将打印"Hello".在终值999输入之后，运行结束时将再次打印"Hello".这是因为没有 END 语句才发生的.DO WHILE … LOOP 终止后，程序流程返回到 LOOP 后的语句，那里 PrintHello 将执行第二次，此时，就会打印出如下的错误信息:

```
Error  3:  RETURN  Without  GOSUB
```

打印该错误信息是由于在LOOP 之后顺序地执行了PrintHello，而不是由GOSUB 调用的结果，因此，不要求有 RETURN.

注意，开始时没有发生语法错误，是因为在程序中END语句并不特别要求.运行期间打印第 3 号错误是因为第二次通过子程序 PrintHell 时，计算机遇到了 RETURN，但是无处返回.

应保证子程序的执行，只能GOSUB 调用，切记主程序中应包括END语句，子程序的标号对应子程序终点的 RETURN.

2. 标号后没有跟冒号.

有时程序员错误地使用分号或空格，而不是冒号:

⋮

```
END
PrintHello                  注意，此处无冒号
    LPRINT "Hello!"
RETURN
```

这种情况下将发生下面的语法错误:

```
Error 456: Underfined label
```

光标也将指向GOSUB 语句的标号PrintHello，表示PrintHello 是没定义的标号，要改正这个错误，只需在编写子程序时在 PrintHello 后加上冒号:

```
PrintHello:
    LPRINT "Hello!"
RETURN
```

§5.5.6 GOSUB 特性总结

在自顶而下模块化程序中应用 GOSUB 的优点

1. 程序易于阅读和维护.
2. 避免了重复编程.
3. 程序逻辑易于跟随.
4. 程序结构易于设计.

编写子程序指南

1. 每个子程序应是一个自成体系的单元，代表一种独立的操作或功能.
2. 保持子程序的单一目的性（例如：两个独立的程序完成两个独立的任务）
3. 保持子程序的短小性.
4. 用自顶而下或层次化方式编写程序.

自我测验(四)

考虑下面的程序

```
AddEven:
    Sum = Sum + EvenNum
    EvenNum = EvenNum + 2
RETURN
```

1. 编一程序调用上面的子程序，计算从2到2000（包括两端点）的所有偶数之和。
2. 编一程序调用上面的子程序，计算从50到3002（包括两个端点）的所有偶数之和。
3. 编一程序调用一个子程序，计算从1到2500（包括两端点）的所有整数之和。
4. 编一程序调用一个子程序，计算N!，即N的阶乘。
   其中，N! = (N) × (N−1) × (N−2) × … ×1（例如：5! = 5×4×3×2×1 = 120)

指出下面程序节选中的错误是什么（5—10)

5.
```
GOSUB XYZ
  ⋮
XYZ
  ⋮
RETURN
```

6.
```
GOSUB Error
  ⋮
Error:
  ⋮
RETURN
```

7.
```
CLS
GOSUB EnterData
DO WHILE CustNo > 0
    LET BalanceDue = BalanceDue + NewCharges - Payments
    LPRINT CustNo, CustName$, BalanceDue
    GOSUB EnterData
LOOP
LPRINT "End of Report"
END
EnterData
PRINT "Please enter Customer Number, Customer Name, "
PRINT "BalanceDue, NewCharges, Payments"
PRINT "When out of data enter negative Customer Number"
```

```
INPUT CustNo, CustName$, BalanceDue, NewCharges, Payments
END SUB
```

8.
```
CLS
Total = 0
GOSUB EnterData
DO  WHILE Score > 0
    LET Total = Total + Score
LOOP
END
EnterData:
    PRINT "Enter Student Number, Test Score"
    INPUT Student, Score
RETURN
```

9.
```
CLS
LET A = 0
LET B = 16
LET C = 22
GOSUB Calculate
END
Calculate:
    LET A = B + C
    PRINT
    PRINT B, C, A
    GOSUB Calculate
RETURN
```

10.
```
CLS
LET B = 16
LET C = 22
GOSUB PrintRoutine
LET A = B + C
PrintRoutine:
    PRINT A
    PRINT B, C
RETURN
END
```

11. 什么时候应用子程序最有好处?

12. 一个子程序能调用另一个子程序吗?

§5.6 本章辅导

§5.6.1 本章小结

Ⅰ. 四种逻辑结构的回顾

A. 顺序

程序是按照其出现的顺序执行。

B. 选择

IF—THEN—ELSE 根据某条件是否满足，选择要执行的指令或指令集。

C. 迭代

根据给定的一个条件或若干个条件是否满足，反复执行一个指令序列.

D. 分情形选择

执行若干个指令中的一个，取决于一个变量或若干变量的内容。

Ⅱ. 迭代方法

A. DO WHILE … LOOP 可实现迭代:

1. 重复执行循环，直到条件不再满足.
2. 如果条件一开始就不满足，循环将不执行，因为条件的测试是在第一次执行循环之前，要首先执行循环，然后再测试条件，则将 DO 写在第一行，而 LOOP WHILE ... 放在循环的最后一行.
3. 可用一个终止值或计数器控制循环.
   a. 如果用计数器，则它必须初始化.
   b. 每执行一次循环，计数器必须增加或减少.
   c. DO WHILE 用来测试计数器.
   d. 当计数器处理不当时，通常出现的问题是死循环. Options命令下的 Keyboard Break 选项应置为“ON”，使得可用 Ctrl −Break 键来终止死循环.

B. DO WHILE的变形

1. WHILE … WEND 可用来代替 DO WHILE … LOOP. 即: WHILE. . . WEND 与 DO WHILE … END 是相同的.
2. DO UNTIL … LOOP 执行循环，直到满足特定的条件.

C. 循环中的EXIT 语句

1. EXIT 语句用来提前退出循环.
2. EXIT LOOP 用来退出DO WHILE, DO UNTIL, 或WHILE … WEND 结构

D. 嵌套迭代

1. 所有的迭代形式均可嵌套
2. 每个DO (或WHILE) 与最近的LOOP 匹配.

Ⅲ FOR … NEXT 语句

A. 当要执行预定次数的循环时，可使用FOR … NEXT语句，在FOR … NEXT 中，每重复一次循环，需对一个变量增加（或减少）一个固定值.

B. FOR … NEXT完成下述工作:

1. 建立一个变量.
2. 增加（或减少）该变量.
3. 指定该变量变化的范围.

C. FOR … NEXT的改进

1. FOR … NEXT中变量的定界符既可是数字，也可以是数值变量.
2. 如果增量不是+1，则需包括STEP分句.
3. FOR … NEXT可以嵌套（注：每个NEXT与最近的FOR 匹配）
4. FOR … NEXT可以包含一个EXIT语句，提前终止循环。

Ⅳ. 应使用哪种迭代方法

A. 当循环执行次数是一个变数时，最好使用DO … LOOP 方式。
B. 当处理执行的次数是一个固定数时，最好应用FOR … NEXT.

Ⅴ. 将自顶而下模块化编程方法引入子程序

A. 一个模块或子程序是用作功能单元并执行特定操作集合的一套指令.
B. 子程序减少了重复性工作.
C. GOSUB语句引起执行一个独立的程序，然后程序流程返回到GOSUB后面的语句.
D. 子程序编写到END语句之后.
E. 每个子程序由RETURN结束，RETURN在程序中占一行.
F. 子程序实现自顶而下、模块化设计.
G. 子程序由一个标号标识．在GOSUB X Y Z 语句中，XYZ 是标号，结构如下所示:

```
GOSUB XYZ
     .
     .
     .
END
XYZ:
     .
     .
     .
RETURN
```

1. 标号在GOSUB语句中给定.
2. 在END语句后面，标号是子程序的首行，它后面跟一冒号.
3. 行号可代替标号，但建立不这样做.

H. GOSUB语句可以嵌套.
I. 常见错误:
1. 在一个子程序中没有RETURN.
2. END 语句后没有编程所有的子程序.

§ 5.6.2　本章自测

1. (T或F) 如果某条件为真执行一套指令，条件是假的，执行另外一套指令，在此情况下应使用循环。
2. (T或F)　FOR 语句中的计数器或变量必须是整数.
3. (T或F) 所有子程序应编写在程序的最后.
4. (T或F) 子程序仅在程序中的一处引用.

指出下面每个 FOR 语句的错误.

5. FOR Ctr = −1 TO −15
6. FOR Ctr = −1 TO 4 STEP −1
7. FOR N$ = 1 TO 10
8. 如果循环中的计数器或变量没有修改，将会发生什么情况?
9. 解释DO WHILE … LOOP 与DO UNTIL … LOOP 之间的区别.
10. 为什么最好应用自顶而下，模块方法设计和编写程序.

§ 5.6.3　实例

1. 编写一个程序:

   (a) 不用子程序;
   (b) 使用子程序;

打印10 个班中的每班期终考试的平均成绩. 每班有20个学生，每个学生都参加每种考试，因此，将有 200 行输入. 假定头 20 行是 1 班，下边 20 行是 2 班，如此下去.

不用子程序的程序

```
'**********************************************************
' This program calculates the class average for 10 classes
' each with 20 students in it  - no subroutines
'**********************************************************
CLS
Let ClassCtr = 1
DO UNTIL ClassCtr > 10
     Let StudNo = 1
     Let Total = 0
     DO UNTIL StudNo > 20
          INPUT Exam
          LET Total = Total + Exam
          LET StudNo = StudNo + 1
     LOOP
     LET Average = Total / 20
     PRINT "Class Average for Class ";ClassCtr;" is ";Average
     ClassCtr = ClassCtr + 1
LOOP
END
```

```
'*************************************************************
'This program prints a class average for 10 classes with
' 20 students per class
'
'
'We do not need an initial read because we know precisely
'how many variables we will be processing
'*************************************************************
LET ClassCtr = 0
DO WHILE ClassCtr < 10
        GOSUB Initialize
        DO WHILE StudNo < 20
                GOSUB InputAndTotal20Exams
        LOOP
        GOSUB ClassAverageModule
LOOP
PRINT "End of Job"
END
'Subroutines go here
Initialize:
        LET StudNo = 0
        LET Total = 0
RETURN
```

2. 打印一个折旧表:

(a) 不用子程序

(b) 用子程序，该子程序采用折旧的直线法.

国家税务部门根据财产的预期寿命，允许削减征税. 折旧或免税的数量可以从税收中推导出来. 因此，公司为它们的很多财产准备了折旧表. 这样，每年的免税或折旧额就可简单地由该表根据相应年限的输入而得到. 任何给定时间财产的价值看作它的"登记价值"，它等于其原始价值减去累计折旧量.

在某财产预期使用寿命的最后期限，它的价值应折旧到处理品或废品的价值，即"保留价值". 例如，假定某个项目（如汽车）的使用寿命是 5 年，如果该车在 5 年底估计价值为 $ 2000，这个数值就叫做废品或保留价值.

有很多方法把折旧额分布在一项财产的整个寿命上，该程序中的应用方法叫做"直线折旧"；即，一个数值等于财产的买价减去它的废品价. 该数值在财产的可折旧期或使用寿命期间内每年削减.

用直线法为指定财产准备一个折旧表时，将使用图5.3 中给出的问题描述. 从当前时期开始，按如下方法计算折旧额:

折旧额 = （购买价−保留价）/使用寿命

交互输入

项目描述(Item $)， 投资(Cost)， 年数(Years)

输入输出样例

```
Please enter Item Description, Cost
and  Number of years to depreciate
? FLOOR PADS, 10.00, 5

ITEM FLOOR PADS
COST 10
Year            Depreciation to date        Remaining Value
 1              2                           8
 2              4                           6
 3              6                           4
 4              8                           2
 5              10                          0
```

图 5.3 实例 2 的问题描述

要求打印一个折旧表，指出年折旧额．例如，如果某项财产预计可使用6年，每年将折旧该财产的 1／6；这是因为用直线折旧法，每个周期的折旧额是相同的．因而，将有 6 个年折旧数字打印在折旧表中．

造出的表指出了''.当前日期到财产的使用寿命结束每年的折旧或免税额．

假定某设备购买价为 $ 1100，废品价为 $ 1000，希望打印一个以年为单位的3年折旧表，因而，将要计算 3 个年折旧数值．

折旧额计算公式如下:

$$\frac{\text{价值}-\text{废品价}}{\text{周期数}}=\frac{11000-100}{3}=3333.33$$

因此，三年中每年的折旧额为 $ 3333.33．用直线折旧法，每年的折旧率是一样的．

输入项目描述、投资和年数，并显示折旧表．下一章将集中讨论在各行中整齐地打印标题和数值的方法．

不用子程序的程序

```
'************************************************************
'    This program calculates the depreciation for an item
'      over its useful life - without subroutines
'************************************************************
CLS
PRINT "Please enter Item Description, Cost"
PRINT  "and  Number of years to depreciate"
INPUT Item$,Cost,Years
YearlyDep = Cost / Years
PRINT
PRINT
PRINT "ITEM ";Item$
PRINT "COST";Cost
PRINT "Year","Depreciation to date","Remaining Value"
FOR   YearCount   = 1 TO Years
     DepDate = YearlyDep * YearCount
     RemVal = Cost - DepDate
     PRINT YearCount,DepDate,,RemVal
NEXT YearCount
END
```

用子程序的程序

```
'************************************************************
'       This program calculates the depreciation of an item
'       over its useful life.  The program uses subroutines.
'************************************************************
CLS
GOSUB EnterData
YearlyDep = Cost / Years
GOSUB PrintHeadings
FOR YearCount = 1 TO Years
        GOSUB ProcessAYear
NEXT YearCount
END
EnterData:
        PRINT "Please enter Item Description, Cost"
        PRINT "and Number of years to depreciate"
        INPUT Item$, Cost, Years
RETURN
PrintHeadings:
        PRINT
        PRINT
        PRINT "Item "; Item$
        PRINT "Cost "; Cost
        PRINT "Year", "Depreciation to date", "Remaining Value"
RETURN
ProcessAYear:
        DepDate = YearlyDep * YearCount
        RemVal = Cost - DepDate
        PRINT YearCount, DepDate, RemValue
RETURN
```

§5.6.4 复习题

1. 一个公司付给其15个营业员（01—15号)每个月＄2500薪水加2%佣金．写一个程序输入营业员号和销售量，计算并打印销售量及薪水和佣金.

2. 打印一个表，计算从1到100的平方和平方根，注意无需输入。
3. 通货膨涨率是按一个项目从一年到下一年价格增长百分比来计算的。编一个程序，用子程序输入通货膨涨率，然后将输入数目未知的项，即 Item $；然后，输入每个项目的上一年价格（Oldprice），计算并打印基于通货膨涨率和旧价格的新价。

§5.6.5 调试练习

指出下面的错误。如果用户愿意，可选用 Trace ON 选项。若选用 Trace ON 选项，则应给每个程序行编号。

1.
```
'This program prints a total for inputted 10 amounts
LET Ctr = 0
LET Total = 0
DO WHILE Ctr <= 10
    INPUT Amt
    LET Total = Total + Amt
LOOP
PRINT "The Total is "; Total
```

2. With the Trace ON option set, indicate the errors in the following:

```
10  FOR A = 1 TO 5
20          FOR B = 3 TO 15 STEP 3
30              PRINT "A = "; A," B = "; B
40          NEXT B
50  GOSUB 30
60  NEXT A
70  PRINT "End of Program"
80  END
```

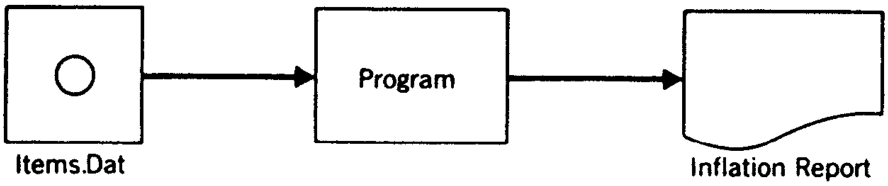
3.
```
 10  GOSUB Initialize
 20  DO  WHILE Cost > 0
 30      GOSUB EnterData
 40      LET SalesTax = Cost * .06
 50      PRINT "Total Due = "; SalesTax + Cost
 60      NoSales = NoSales + 1
 70      TotalSales = TotalSales + Cost
 80  LOOP
 90  PRINT "Number of Sales = "; NoSales
100  PRINT "Total Sales Amount ; "; TotalSales
110  PRINT "Average Sale = "; TotalSales / NoSales

120  PRINT "End of Report"
130  END
140  Initialize:
150      NoSales = 0
160      TotalSales = 0
170  EnterData:
180      PRINT "Please enter the item sale price
190      PRINT "When there is no more data enter"
200      PRINT "a number less than zero."
210      INPUT Cost
220  RETURN
```

```
4. LET X = 0
   DO  WHILE X > 10
       LET Y = X * 2
       PRINT "X = "; X
       PRINT "Y = ":Y
       LET X = X + 1
   LOOP
   END
```

§ 5.6.6　编程题

1. 本程序的问题描述见图5.4:

系统流程图



Items.Dat 记录格式

ItemNo, ItemDescription$, UnitCost

输出样例

```
                              I N F L A T I O N   R E P O R T

ITEM NUMBER:  11111
ITEM DESCRIPTION:   TEST
ITEM COST:     $1.00

                    YEAR      INFLATION RATE     ITEM COST WITH INFLATION
                     1             08%                      1.08
                     2             08%                      1.16
                     3             08%                      1.25
                     4             08%                      1.35
                     5             08%                      1.45
                     6             06%                      1.53
                     7             06%                      1.62
                     8             06%                      1.71
                     9             06%                      1.81
                    10             06%                      1.91
ITEM NUMBER:  12345
ITEM DESCRIPTION:   RAKE
ITEM COST:     $12.45

                    YEAR      INFLATION RATE     ITEM COST WITH INFLATION
                     1             08%                     13.44
                     2             08%                     14.51
                     3             08%                     15.67
                     4             08%                     16.92
                     5             08%                     18.27
                     6             06%                     19.36
                     7             06%                     20.52
                     8             06%                     21.75
                     9             06%                     23.05
                    10             06%                     24.43
```

输入样例

```
11111TEST                00100
12345RAKE                01245
12346SHOVEL              02599
12347AX                  02088
23456SNOW SHOVEL         01499
43256HOE                 03999
```

图 5.4 编程题 1 问题描述

给出每一项的初始值，希望打印一个表，给出每一项的跨度为10年的通货膨涨额，假设前5年通货膨涨率为8%，而后5年的通货膨涨率为6%，累计通货膨涨的效果。

```
ITEM COST: $1.00

YEAR 1      1.08      (1 x 1.08)
YEAR 2      1.08^2    (1.08 x 1.08)
  .
  .
  .
YEAR 5      1.08^5
YEAR 6      1.08^5 x 1.06
  .
  .
  .
YEAR 10     1.08^5 x 1.06^5
```

2. 编一程序并绘图表示不同星球与地球重量的比较。其问题描述参见图5.5.

注:

(a) 采用下表:

行星	占地球重量的百分比
Mars	40
Venus	85
Jupiter	264
Saturn	116

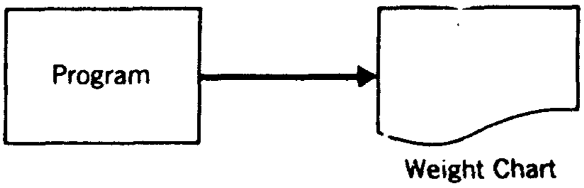
(b) 不需要输入

(c) 图形应包括地球重量，范围从 50 到 250，增量 50，

3. 输入本金(Principal)，利率(Intestest)和投资年限(NoOfYears)，对每个输入打印如下项:

```
NO OF YEARS    PRINCIPAL    INTEREST DURING    ACCRUED INTEREST
                            THIS PERIOD
     1
     .
     .
     .
     N
```

系统流程图



输出格式

WEIGHT CHART

EARTH	MARS	VENUS	JUPITER	SATURN
50	XXX.X	XXX.X	XXX.X	XXX.X
100	↓	↓	↓	↓
150				
200				
250				

图 5.5　编程题 2 的问题描述

4. 输入每组输入的标题信息如下:

   CLASS NO:

   NO OF STUDENTS PER CLASS:

   这些行后的输入行与每班的学生数量相等，每个班重复打印:

   CLASS NO:　　　　　　AVERAGE GRADE

第　三　部　分
打　　印

本 部 分 内 容

第六章　利用输入文件或键入数据
显示和打印设计良好的报表
第七章　控制中断过程：自顶向下模块化方法

第六章　利用输入文件或键入数据显示和打印设计良好的报表

本章要求

用户应了解如下内容:

1. 常用作打印报表程序输入的顺序文件;
2. 打印输出的主要特点;
3. 设计界面友好打印报表的方法;
4. 增强打印输出易读性可执行的几种编辑方法;
5. 打印报表类型.

§6.1　文件处理导论

§6.1.1　什么是文件?

至此，各程序的输入数据皆从键盘输入，然后输出显示在屏幕上或打印机上印出. 这叫做交互处理(interactive processing). 对于从键盘上键入的每一行或多行交互式输入，程序可立即产生某种形式的输出.

然而，打印报表及其它类型的输出常常由输入文件产生. 文件(file)是一个与特定应用相关的数据集合，如应收账文件、工资单文件、应付账文件等等. 文件通常存在软盘或硬盘上，随后通过批处理(batch processing)产生大量输出. 这即意味着整个文件通常定期进行一次性处理. 例如，我们可以读入一个工资单文件来生成一组薪金支票或者打印一份周薪水报表.

为便于存储和检索，磁盘文件中的数据有三种组织方式，即顺序、随机和二进制. 本章将简述建立顺序文件的方法以及按批方式读入这些文件生成打印报表的方法. 第十三章将详细介绍顺序及随机类文件的处理方法.

顺序文件是这样一种文件，即其中的每一单元按顺序读入，从第一个开始，随后读第二个等等. 例如，若要存取或读一顺序文件的第 986 个记录，必须先读过前 985 个记录.

§6.1.2　文件中的记录

所有文件，不管其采用何种组织方式，皆由记录组成. 记录(record)是与文件中一条

输入有关的信息单位．例如，工资单文件由雇员记录组成；应收账文件由客户记录组成等等．记录内有字段(field)，字段是由单个字符组成的数据项，例如，一条雇员记录可有社会保险号字段、雇员姓名字段、工资字段等．顺序文件中的字段可由数值或串变量组成．

这样，字段由字符组成，记录由字段组成，而文件又由记录组成．如图 6.1 所示．

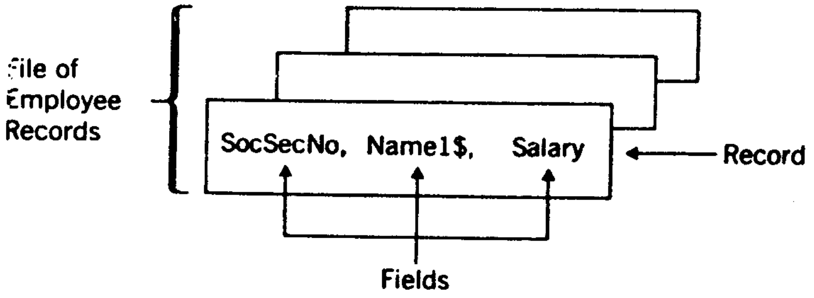


图 6.1 字段、记录和文件

§ 6.1.3 建立顺序文件

§ 6.1.3.1 如何建立顺序文件

磁盘文件由程序生成，用户所遇到的大部分文件是由 Turbo Basic 程序、文件本编辑器及称作数据库管理系统(data base management systems) 的软件包生成的．此处将着重介绍用 Turbo Basic 程序建立顺序文件的方法，随后再写一些 Turbo Basic 程序读这些文件并打印输出．

先写一个简单程序，使用户可键入数据，建立一个顺序存货文件．此文件由数据记录组成，每条记录包括 PartNo、Description $ 、UnitPrice 及 QtyOnHand 字段．变量由用户从键盘输入，每个变量作为单个字段存储，而每一部分的所有字段之集合则作为单个记录存在磁盘文件中．输入变量的数据录入过程应包括给用户的提示：

```
PRINT "Enter Part No:"
INPUT PartNo
PRINT "Enter Item Description:"
INPUT Description$
PRINT "Enter Unit Price:"
INPUT UnitPrice
PRINT "Enter Quantity On Hand:"
INPUT QtyOnHand
```

§ 6.1.3.2 建立文件时应确认数据

人们有时会出错，数据录入员也不例外．接纳输入数据并建立文件所有程序皆应包括一些例程来检验输入的准确性，以便最大限度地减少错误．第八章将讨论较为高级的数据确认过程．这里我们感兴趣的只是建立一个顺序文件来打印报表，不过，应当记住建立这些文件的"真实"程序还要确认数据，以把出错率降至最低．

§ 6.1.3.3 OPEN 语句

待建立文件必须首先打开．一个 OPEN 语句执行以下操作:

a. 为读、写准备磁盘;
b. 标识待建数据文件名;
c. 指定文件存取方式．磁盘文件可按顺序方式组织，即记录总是按顺序读．在第十三章，用户将会看到磁盘文件也可按随机方式组织，这样，记录则不必按顺序读;
d. 指明文件是用作输入还是输出．一个作为输出建立的文件可用作另一程序的输入．本节将建立一个用作输出的存货文件，在下一节，该文件将被用作打印程序的输入．

顺序文件 OPEN 语句格式为:

```
OPEN filespec [FOR mode] AS [#] filenum [LEN = record size]
```

或:

```
OPEN mode string, [#] filenum, filespec [, record size]
```

这些格式成分相同，只不过次序不同．下面详细讨论每一成分:

输入项	含义
Mode	用下述之一: OUTPUT — 指定一个待写顺序文件. INPUT — 指定一个待读顺序文件. APPEND — 指定一个待附加顺序文件(APPEND 用于往一个顺序文件结尾增加记录).
Mode string	这是一个串表达式，对于顺序文件，它的第一个(通常只有一个)字符是下列之一: O — 指定顺序输出方式. I — 指定顺序输入方式. A — 为往一个文件末端增加记录指定顺序输出方式.(此外，还允许 R 和 B 方式，R 代表随机方式，B代表二进制方式.将在第十三章讨论).
Filenum	这是个整数值.如处理若干文件，一个为#1. 另一个为#2，等等.如一个输出文件作为#1 打开，则往该文件写时必须用#1.如一个输入文件作为#1 打开，则从该文件读入记录时必须用#1.
Filespec	这是一个指定所用文件名字的串表达式(1 至 8 个字符，后跟一可选扩展名，扩展名最多为 3 个字符，扩展名与文件名用一个小数点隔开).整个名字(包括扩展名)应当用引号括起来.
Record size	这是一个从 1 至 32767 的整数表达式.缺省记录尺寸为 128 字节，即如果省略记录尺寸，则计算机假定每个记录最多包含 128 个字符.

上面指定的两种 OPEN 语句格式给出相同的信息，但第一种较为友好.

例:

```
OPEN "Inventry.Dat" FOR OUTPUT AS #1
```

该 OPEN 语句也可写作:

```
OPEN "O", #1, "Inventry.Dat"
```

如果某个文件在一个程序中作为#1 打开，则它在该程序中所有其它输入／输出操作中都必须称为#1. 另一方面，第二个程序可称之为#2，只要在整个程序中保持一致即可。

上例中的文件名"Inventry.Dat"必须包含在引号内. 编译程序假定文件将建在缺省驱动器上，如需用一不同的驱动器，可用驱动器名加冒号作为文件名前缀. 注意到 Inventory 中的字母 o 省略去了，因为文件名最多可包含八个字母.

例:

```
"B:Inventry.Dat"
```

文件名可有扩展名，如上面的·Dat 或其它别的扩展名，但扩展名不是必须的. 本书中所有数据文件都用·Dat 作扩展名.

§ 6.1.3.4 WRITE 语句

往一个顺序文件输出数据，用 WRITE 语句:

```
WRITE #filenum, expression list
```

filenum 与输出文件的 OPEN 语句中用到的号相同. 这样，如输出文件作为#3 打开，则往该文件 WRITE(写)，也用#3:

```
OPEN "Inventry.Dat" FOR OUTPUT AS #3
•
•
•
WRITE #3, PartNo, Description$, UnitPrice, QtyOnHand
```

输出是按表达式表写至文件的，如下所述:

1. 表达式表中的条目(即字段)间插入逗号.
2. 如要输出串数据，如上面的Description $，则每条记录中的值两边会自动加

上引号，以将之标别为串.

3. 顺序文件中的每条记录，依其字段尺寸，可有不同数目的字符，字段间用逗号隔开.
4. 每条记录以一个回车或换行符结束.

下面给出输出至磁盘文件的记录的一组设计准则:

设计磁盘文件准则

1. 每条记录的首字段或变量应当是一个关键字段(key field)，用以标识记录. 依记录类型的不同，关键字段可以是社会保险号、交易号、客户号等.
2. 所建记录应尽可能简洁，以节省磁盘空间，例如，在一存货文件中，零件号可用于替代项目描述，因为零件号字段需要较少的字符.
3. 应避免字段中用逗号、美元符、白空此类编辑等. 注意到这些符号在数值变量中不允许的.
4. 用编码字段替代描述字段. 例如，可用代码1、2、3或"M"、"S"、"D"指示婚姻状况，来代替较长的描述项，如"Married"(已婚)、"Single"(单身)或"Divorced"(离异).
5. 应特别注意把产生错误输出的可能性降至最低(第八章将着重阐述用于此目的的错误控制过程之类型).

§6.1.3.5 CLOSE 语句

在中止一个建立或读文件的程序之前，应当 CLOSE(关闭)文件. CLOSE 语句的格式为:

```
CLOSE [[#] filenum [, [#] filenum] ... ]
```

例:

`CLOSE #1`	Closes the file opened as #1
`CLOSE #1, #2`	Closes the files opened as #1 and #2
`CLOSE`	Closes all files that have been opened

§ 6.1.3.6 建立顺序文件示例

下面是一个可使数据录入员建立顺序存货清单文件的完整程序。假定数据按 PartNo 序列输入。

EDIT 窗口

```
'This program creates a sequential inventory file
CLS
OPEN "Inventry.Dat" FOR OUTPUT AS #1
GOSUB PromptAndGetInput
DO WHILE PartNo < 9999
        WRITE #1, PartNo,PartDescription$, UnitPrice,QtyOnHand
        GOSUB PromptAndGetInput
LOOP
CLOSE #1
END
PromptAndGetInput:
        PRINT "Enter Part No:"
        INPUT PartNo
        PRINT "Enter Part Description:"
        INPUT PartDescription$
        PRINT "Enter Unit Price:"
        INPUT UnitPrice
        PRINT "Enter Quantity On Hand:"
        INPUT QtyOnHand
RETURN
```

RUN 窗口

```
Enter Part No:
? 001
Enter Part Description:
? SHIRTS
Enter Unit Price:
? 35.00
Enter Quantity On Hand:
? 40
Enter Part No:
? 002
Enter Part Description:
? TIES
Enter Unit Price:
? 15.50
Enter Quantity On Hand:
? 120
```

```
Enter Part No:
? 003
Enter Part Description:
? SLACKS
Enter Unit Price:
? 75.00
Enter Quantity On Hand:
? 75
Enter Part No:
? 004
Enter Part Description:
? SUITS
Enter Unit Price:
? 275.00
Enter Quantity On Hand:
? 50
Enter Part No:
? 005
Enter Part Description:
? SOCKS
Enter Unit Price:
? 8.50
Enter Quantity On Hand:
? 120
Enter Part No:
? 006
Enter Part Description:
? JEANS
Enter Unit Price:
? 30.00
Enter Quantity On Hand:
? 60
Enter Part No:
? 9999
Enter Part Description:
? STOP
Enter Unit Price:
? 9999
Enter Quantity On Hand:
? 9999
```

Inventry.Dat 文件

```
1,"SHIRTS",35,40
2,"TIES",15.5,120
3,"SLACKS",75,75
4,"SUITS",275,50
5,"SOCKS",8.5,120
6,"JEANS",30,60
```

上述程序输入过程中用了一个 GOSUB 以避免重复，并使程序模块化。

上面方法中要求用户给 PartNo 输入一个结尾值 9999 以表明任务结束。不过，这只在真正的 PartNo 没有 9999 这个值时方可。下述方法不用初始读入，而是问用户是否想继续，如果有更多的输入，则键入 Y，如果应当中止运行，则键入 N：

EDIT 窗口

```
'This program creates an inventory file
'  and prompts for more input
'There is no initial read
OPEN "Inv2.Dat" FOR OUTPUT AS #1
CLS
MoreInput$ = "Y"
DO WHILE MoreInput$ = "Y"
        PRINT "Enter Part No:"
        INPUT PartNo
        PRINT "Enter Part Description:"
        INPUT PartDescription$
        PRINT "Enter Unit Price:"
        INPUT UnitPrice
        PRINT "Enter Quantity On Hand:"
        INPUT QtyOnHand
        WRITE #1, PartNo,PartDescription$, UnitPrice, QtyOnHand
        PRINT "Is There More Input (Y/N)?:"
        INPUT MoreInput$
LOOP
CLOSE #1
END
```

RUN 窗口

```
Enter Part No:
? 001
Enter Part Description:
? SHIRTS
Enter Unit Price:
? 35.00
Enter Quantity On Hand:
? 40
Is There More Input (Y/N)?:
? Y
Enter Part No:
? 002
Enter Part Description:
? TIES
Enter Unit Price:
? 15.50
Enter Quantity On Hand:
? 120
Is There More Input (Y/N)?:
? Y
```

```
Enter Part No:
? 003
Enter Part Description:
? SLACKS
Enter Unit Price:
? 75.00
Enter Quantity On Hand:
? 75
Is There More Input (Y/N)?:

? Y
Enter Part No:
? 004
Enter Part Description:
? SUITS
Enter Unit Price:
? 275.00
Enter Quantity On Hand:
? 50
Is There More Input (Y/N)?:
? Y
Enter Part No:
? 005
Enter Part Description:
? SOCKS
Enter Unit Price:
? 8.50
Enter Quantity On Hand:
? 120
Is There More Input (Y/N)?:
? Y
Enter Part No:
? 006
Enter Part Description:
? JEANS
Enter Unit Price:
? 30.00
Enter Quantity On Hand:
? 60
Is There More Input (Y/N)?:
? N
```

Inv2.Dat 文件

```
1,"SHIRTS",35,40
2,"TIES",15.5,120
3,"SLACKS",75,75
4,"SUITS",275,50
5,"SOCKS",8.5,120
6,"JEANS",30,60
```

上述两种建立顺序文件的方法都是可以接受的。

§ 6.1.4 从顺序文件读入记录

用 WRITE 语句可以输出任意多个存货清单记录．建立输出输序文件所用格式与其后从文件读入记录所用格式相同．在打印程序中从上述文件读入每条记录，可以如此编程:

```
OPEN "Inventry.Dat" FOR INPUT AS #1
.
.
.
INPUT #1, PartNo, Description$, UnitPrice, QtyOnHand
```

从此文件读入记录并打印输出(即生成某种形式的输出)的程序格式如下:

```
OPEN "Inventry.Dat" FOR INPUT AS #1
DO  WHILE NOT EOF(1)
    INPUT #1, PartNo, Description$, UnitPrice, QtyOnHand
    .
    .                   'Process each record
    .
LOOP
CLOSE #1
END
```

DO WHILE NOT EOF(1)意指文件号 1 内的输入应进行处理，直到不再有输入记录为止．这即是说，EOF(1)对文件#1 测试文件结束条件，这样，当处理输入文件时，文件结尾不再有结尾值，而是测试一个 EOF 即文件结束条件，在读人最后一个记录时，该条件满足．如果文件号 1 内有 10 条输入记录，则当第 10 条记录被读入后，EOF(1)条件满足．这样，读 10 条记录，DO WHILE…LOOP 将执行 10 次，INPUT 语句必须出现在循环的开头，以确保处理第 10 条记录．这即是说，DO WHILE…LOOP 的 INPOT 语句第 10 次执行时应使第 10 条记录读入，且使条件 EOF 满足．但由于 EOF(1)在循环结束前不再测试，所以第 10 条记录可被正确处理，即需处理最后一条输入记录，并随后测试 EOF:

```
       LOOP
Read Record     Process

     1             1
     .             .
     .             .
     .             .
    10          <—— EOF is met when this record is read (The last record
                    must be processed before we terminate the loop)
```

至此，从一顺序文件读入时，用户不需用初始读入语句．如省略初始读入，则在 DO WHILE…LOOP 的开头要在 INPUT 语句，这样可在测试 EOF 条件前处理完最后一条

记录。处理顺序文件的另一种方法是用初始读入，如下所示:

```
CLS
OPEN "Inventry.Dat" FOR INPUT AS #1
GOSUB ReadModule
MoreRecords$ = "Yes"
DO  WHILE MoreRecords$ = "Yes"
         .
         .
         .
    GOSUB ReadModule
LOOP
CLOSE #1
END
ReadModule:
    IF  NOT EOF(1) THEN
        INPUT #1, PartNo, Description$, UnitPrice, QtyOnHand
    ELSE
        MoreRecords$ = "No"
    END IF
RETURN
```

与从键盘录入数据不同，从输入文件读入记录的所有程序可以用上述两种格式中的任一种。

自我测验(一)

下列语句错在哪里:

1. INPOT #1 SocSecNo, Name$, Salary
2. DO WHILE EOF

   ⋮

3. OPEN "O", #1, Payroll.Dat
4. WRITE #1, "Employee, Dat"
5. (T或F) 每条记录的首字段就是关键字段.
6. (T或F) 我们用结尾值检测文件结束.
7. (T或F) 建立文件前，需要先OPEN(打开)一个文件.
8. 磁盘文件数据的三种组织方式是_______、_______与_______.
9. 从顺序文件读入时，如在DO WHILE…LOOP(头、尾)有INPUT语句时，(需要、不需要)初始读入。
10. 文件是_______的集合。
11. 将数据输出至一个顺序文件，用_______语句。
12. 从一个顺序文件读入数据，用_______语句。
13. 请解释交互处理与批处理的差别。

14. 请解释存取一个顺序文件第47条记录的方法.

15. 文件、记录与字段有什么关系?

§6.2 打印输出的特点

§6.2.1 显示与打印输出

用户已经注意到，输出可以从键盘交互输入的数据产生，如前面几章即是如此；也可从输入磁盘文件产生，如上节建立的那一类文件.

从文件或键盘录入的数据可产生两种主要的输出形式:

(1) 输出显示在屏幕上;

(2) 输出从打印机上输出.

§6.2.1.1 显示输出

显示输出主要用于:

a. 回答查询. 用户可从一文件索取信息，计算机对该需求响应，在屏幕上显示出所需信息.

例:

下面这个程序读入一个输入文件并显示出它所包含的记录总数.

EDIT 窗口

```
'This program counts records in the Inventry.dat file
CLS
OPEN "Inventry.Dat" FOR INPUT AS #1
RecordCounter = 0
DO WHILE NOT EOF(1)
        INPUT #1, PartNo, Description$, UnitPrice, QtyOnHand
        RecordCounter = RecordCounter + 1
LOOP
PRINT "The number of records in the file is "; RecordCounter
CLOSE #1
END
```

RUN 窗口

```
The number of records in the file is 6
```

b. 交互输出。用户输入一系列变量，计算机基于这些数据进行某些处理，并立即给出输出结果。下面程序中，每行输入三个成绩，每个学生一个。计算机在下一行显示出平均值。

EDIT 窗口

```
  This program reads three exams per student and displays an average
CLS
PRINT "Enter three exams separated by commas"
PRINT "Enter 999,999,999 when done"
INPUT Exam1, Exam2, Exam3
DO WHILE Exam1 < 101
        Average = (Exam1 + Exam2 + Exam3) / 3
        PRINT "The Average is "; Average
        INPUT Exam1, Exam2, Exam3
LOOP
PRINT "End of Job"
END
```

RUN 窗口

```
Enter three exams separated by commas
Enter 999,999,999 when done
? 80,90,70
The Average is  80
? 85,95,90
The Average is  90
? 62,66,70
The Average is  66
? 999,999,999
End of Job
```

§6.2.1.2 打印输出

另一方面，打印报表是正式文档，可以给用户提供需要保存和加工的信息的"硬拷贝"，这通常用磁盘文件作为输入生成，但有些打印报表用从键盘录入的数据作为输入。

由于计算机生成的显示和报表是供人阅读的。所以必须清晰、简洁并易于理解。设计磁盘数据文件时，简明性和效率是压倒一切的因素，而设计显示或打印输出时，与用户友好才是人们主要关心的。

§6.2.2 打印和显示输出所用的方法

本节将要介绍制作和设计打印报表所用方法(大部分方法也适用于输出显示)。先介绍这些特点，然后再用 Turbo Basic 示范。

§ 6.2.2.1 打印数据必须编辑

磁盘文件的一个记录，举个例子，可能有两个这样的数量字段：450 及 38726.58．尽管这些字段在磁盘文件内是合适的．但打印的报表应当包含这些信息编辑后的形式，这样对用户更有意义．例如．$ 450.00 与 $ 38,726.58 是罗列数据的较为清晰的方法．

编辑(Editing)定义为使数据字段简洁、清晰且更加有用的一种过程．本章将着重介绍对数值和串变量可进行编辑的类型．

§ 6.2.2.2 表格必须留出行空

打印和显示输出的诸行必须适当地留出行空，以便于阅读．某些行间必须空一行，另一些行间要空两行，诸如此类．此外，打印输出的各页顶部和底部应有边白，这要求在所编的程序中判断一页的结束，然后将下一行信息转至新页．

§ 6.2.2.3 必须打印标题

标题信息对于打印信息的明晰表述事关重要，每个报表通常有一个表头(report heading)，它指出报表的题名，如果是份多页报表，还可能有日期和页号．报表也在栏目标题，用于标识待打印字段或变量．

§ 6.2.2.4 数据必须对齐

报表打印输出时，各个变量并不象在磁盘文件中那样彼此邻接或是用逗号隔开，而是在整个页面上均匀、整齐地隔开，这样更容易解释打印输出．用户可以用 Printer Spacing Chart(打印格式表)来规划输出设计以便适当对齐所有各行．参见图 6.2 中的 Printer Spacing Chart 示例；其中包括表头、栏目标题以及数据(明细行)典型的对齐和留空方式．

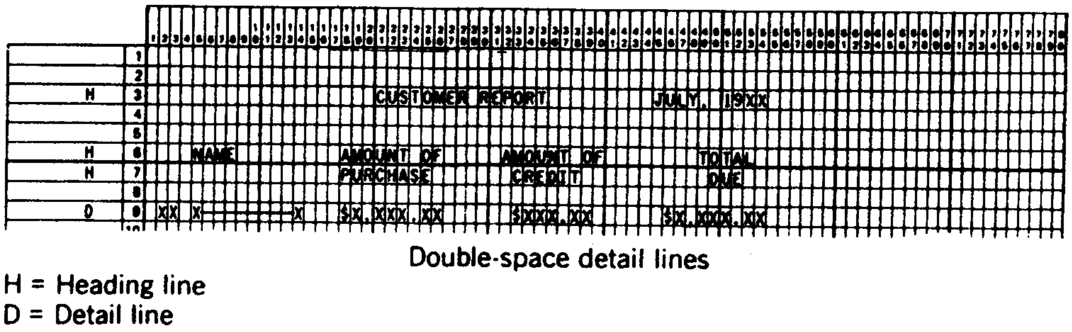


图 6.2 **Printer Spacing Chart 示例**

图 6.3 给出了文件数据可能实际打印过程的原理图。输出将依据图 6.2 中 PrinterSpacing Chart 给出的规范打印。

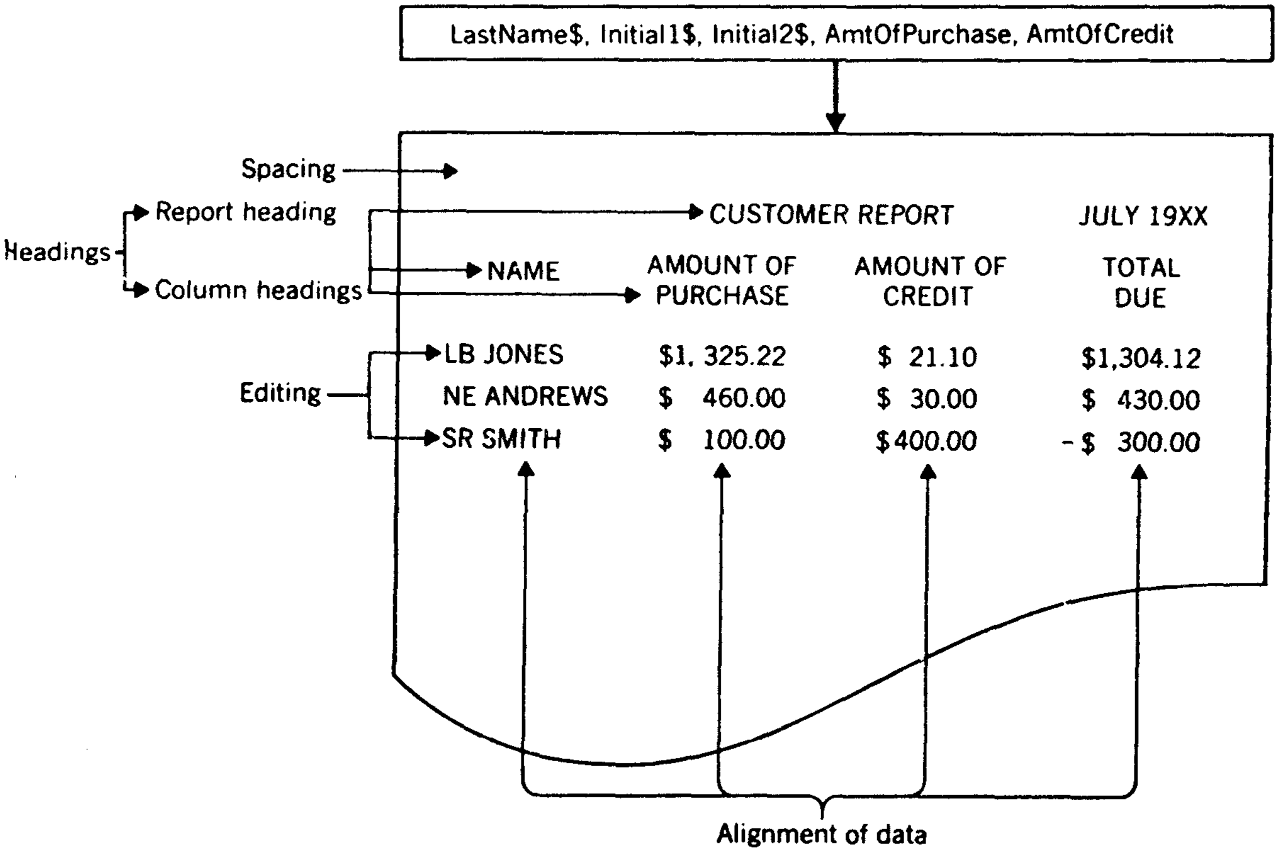


图 6.3　文件数据可能的打印过程原理图

由于打印信息具有不同于其它类型输出的特点，本章将把打印报表作为单独的论题来学习。

§ 6.3　编辑功能

§ 6.3.1　概述

本节将讨论下列编辑功能:

编辑功能

对于数值变量或字段有:

1. 抑制先行零。

2. 不管变量中十进制数字的数目，在小数点右边打印指定数目的数字.
3. 打印美元符及逗号.
4. 打印用于核对保护的星号(*).
5. 在合适的位置(数的左边或右边)打印加、减号.

对于串变量或字段有:

不管串变量的大小，在输出行上打印指定数量的字符.

上述前五个编辑功能只能用于数值变量式域，最后一个编辑功能则用于串变量.考虑下面这个例子:

Edit

```
INPUT Amt
DO WHILE Amt <> 9
   LPRINT Amt
   INPUT Amt
LOOP
END
```

Run

```
? 12.75
? 0
? 125000
```

打印页面上的输出结果为:

```
12.75
0
125000
```

而如按下列形式打印，则结果较易阅读.

```
$      12.75
$       0.00
$ 125,000.00
```

在后一种情况下，请注意:

1. 各数按小数点对齐.
2. 每个值前有一个美元符.
3. 最后一个数中插入了逗号，以增强可读性.

完成此类编辑，可用 LPRINT USING 语句打印输出，或用 PRINT USING 语句显示输出.

§ 6.3.2 LPRINT USING 和 PRINT USING 语句

§ 6.3.2.1 格式

LPRINT USING 和 PRINT USING 语句的两个主要用途如下:

> **LPRINT USING 和 PRINT USING 语句的用途**
>
> 1. 适当对齐结果。
> 2: 编辑结果。
>   在此情形下，编辑(editing)意指加入诸如美元符、逗号和星号之类的编辑符，以增加结果可读性。

LPRINT USING 与 PRINT USING 的格式为:

> { *LPRINT* / *PRINT* } USING 格式字符串表达式表[;]

格式串是一个串常量或变量，用于描述表达式表格式编排的方式。

表达式表由待打印串或数值变量，如果在一行上要打印若干个变量或字段，可用逗号、空格或分号隔开。

格式串包含在双引号("")内，后面带一个分号，分号后是待编排格式的变量。

例:

格式化其后的变量

```
PRINT USING "######"; Amt
```

要格式化的变量

注意到 PRINT USING 语句可以用与 LPRINT USING 语句相同的格式串，两者的区别在于: 用 PRINT USING 时，输出将显示出来，而不打印。

§ 6.3.2.2 编辑数值字段

§ 6.3.2.2.1 对齐数值变量内容

LPRNT USING 或 PRINT USING 语句的格式串中用到的符号"#"表明数值变量将按右齐方式打印．右齐的输出打印时按最右边即个位数位置对齐；例如，87、387 和 2826 三个数打印出来是:

```
  87
 387
2826
```

格式串中的符号"#"不仅把数据按右对齐，而且将先行即最左边的所有零填充为空白．这样，0026、087 与 382 打印为:

```
 26
 87
382
```

这种编辑可确保所有数据打印时按个位数位置适当对齐，并且抑制先行零的出现．另外，LPRINT USING 与 PRINT USING 中的表达式与变量用一个分号隔开．看下面例子:

```
LPRINT  USING  "######"; Amt
```

Amt 的值打印如下:

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
1234	"######"	1234
26	"######"	26
374	"######"	374
26873	"######"	26873
987261	"######"	987261
7522117	"######"	%7522117

如果不用 PRINT USING 语句，输出将按左齐方式打印．上述清单中标有"Value of Amt"的那一栏数据是按左齐方式列出的，不过，按右齐方式打印数更为恰当，这样每个数的个位数字打印在同样的相对位置上，十位数字也打印在同样的相对位置上，等等．用户可以用 LPRINT USINT 或 PRINT USING 语句得到右齐的输出．

还应注意到，LPRINT USING 与 PRINT USING 的规则相同．格式串"######"表明变量值可以是从 1 到 6 个数字的任何数．如果 Amt 的某个值超过了 6 个数字，如上例中最后一个数 7522117，则数前面将打印一个%，%表明变量相对于指定格式串而言太长了．

用户可以用相同的格式串编辑若干变量，这只要用：

```
PRINT USING "#######"; Amt1; Amt2; Amt3
```

即可。

§ 6.3.2.2.2 将字段沿小数位对齐

如果待打印数中有小数点，可在格式串中包含小数点，后面再跟上要打印小数位的数目：

```
LPRINT USING "#######.##"; Amt
```

考虑如下例子：

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
12.75	"#######.##"	12.75
8.60	"#######.##"	8.60
125000.50	"#######.##"	125000.50
4	"#######.##"	4.00
4.1	"#######.##"	4.10
14.7685	"#######.##"	14.7685
87.67	"#######.##"	87.67
-6.48	"#######.##"	-6.48
-.25	"#######.##"	-.25

首先，格式串使数右对齐，其次，每个变量打印结果恰有两个小数位。

表达式表中的变量可以有多达六位整数的任意数目的小数位，这就是说，变量的整数位的数目不应超过格式串前"#"的数目。在上例中，如果变量中实际存储的整数位数超过六，则输出结果前会有%，表明出现长度错误，这意味着变量的整数位多于格式化输出串。

但变量本身小数位(即小数点右边的位置)并不受限制。如果变量中小数位的数目比格式串中指定的大，并不认为出错。假定 Amt 有三个或三个以上小数位，带"##"的 PRINT USING 或 LPRINT USING 语句意味着结果将会四舍五入至两位小数，而不是截至两位小数。这样，如 Amt 值为 12.755，而程序为 LPRINT USING "####"，Amt 打印为 12.76。如果 Amt 的小数位小于两个，则其后补零，即如果用上述格式串，则 4 打印为 4.00，而 4.1 打印为 4.10。如果 Amt 只有小数位，如上例中最后一个(·25)，打印时前面带一个先行零(0.25)。

打印计算机只能近似的小数

用户可能记得，如第三章所述，计算机将小数表达为：

$$\frac{a}{2}+\frac{b}{4}+\frac{c}{8}+\frac{d}{16}+\frac{e}{32}\cdots$$

如果小数只能用这种形式近似，则存储时需用大量数字来近似指定值。实际上，通常用 16 个十进制数字来表示计算机只能近似的数，这些数叫作双精度数(double—precision mumber)，用以较为精确的反映变量或字段的内容。

看下面例子：

Edit
```
INPUT Amt
PRINT Amt
```

Run
```
? 3.17
3.170000076293945
```

尽管用户输入的 3.17，但计算机必须按近似形式存储该数，这是用 16 个十进制数字实现的。3.170000076293945 是计算机对 3.17 的最佳近似，这个数叫双精度数．为确保这些小数总是打印两个小数位，可用 LPRIN 或 PRINT USING 语句，如下所示：

```
INPUT Amt
PRINT USING #.##
```

```
? 3.17
3.17
```

从此，用户应记住：必须由计算机近似的任意小数在打印时，应当用 PRINT 或 LPRINT USING 来指出要打印的十进制数字的数目。否则，双精度型的全部 16 个十进制数字都会输出。只有 0.25、0.50、0.75 这些小数打印时与输入的完全相同，因为它们可以准确地表示为$\frac{a}{2}+\frac{b}{4}+\frac{c}{8}+\cdots$，其它的小数只能由计算机近似，打印时通常要用 PRINT 或 LPRINT USING 来指明要打印的小数位数目。

B18

§ 6.3.2.2.3 在数值字段中插入逗号

如想在打印或显示数据时用一个或多个逗号将整数隔开，只需在 PRINT USING 或 LPRINT USING 语句的格式中包含逗号。所有数据将依然按小数点对齐，且是右齐的。

```
LPRINT USING "#,###,###.##"; Amt
```

例如，如果 Amt 的值为 1052000.50，则打印为：1,052,000.50。只有当 Amt 的值有足够的有效位(即非零整数位)，逗号才会打印出来，否则，逗号将被抑制，即代之以空白。这样，如 Amt 的值为 532.50，打印时不会有逗号，而打印为 bbbbbb532.50，此处 b代表

一个空格。每一行打印或显示数据按小数点对齐。

下面为打印逗号的 LPRINT USING 或 PRINT USING 示例:

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
4872.83	"###,###.##"	4,872.83
873586.487	"###,###.##"	873,586.49
287.05	"###,###.##"	287.05
31.621	"###,###.##"	31.62
2876543.92	"###,###.##"	%2876,543.92

注意: 最后一例中的%表明变量相对于格式串而言太长。格式串允许至多六位整数(即小数点左边可以有 6 位)。

当变量的 LPRINT (或 PRINT) USING 的格式串长时，前面将打印一个%，表明出错或溢出。

另一种在每三个一组的标准位置插入逗号的方法是直接在小数点前放置一个逗号，其结果是在每三个一组的适当位置放置逗号.

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
23872.61	"#######,.##"	23872.61
277873.77	"#######,.##"	277873.77
123.00	"#######,.##"	123.00
1234567.00	"#######,.##"	%1234567

双精度数

单精度数值变量或字段只能精确到六位. 因此，如果想存储一个七位以上数，应将此变量设为双精度数. 双精度数的变量名后带"#"，例如，Amt#即是一双精度变量。

有些程序员对所有数值变量都用双精度型，以确保这些数对大多数类型的处理足够大. 作为一条准则，本书建议: 如果一个数可能超过六位，则用双精度.

例:

```
1. INPUT  Amt#
2. Total#  =  12345678
```

§ 6.3.2.2.4　在数值字段中插入美元符

如想在打印字段内容时带一个美元符，可在 LPRIT 或 PRINT USING 语句中包含一个美元符：

```
LPRINT USING "$#,###.##"; Amt
```

如果 Amt 有一个值为 3546.23，则打印为 $ 3,546.23。

考虑下面的例子：

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
28.73	"$#,###.##"	$　　28.73
4827.26	"$#,###.##"	$ 4,827.26
927.16	"$#,###.##"	$　927.16
0.06	"$#,###.##"	$　　 0.06

§ 6.3.2.2.5　使用浮动美元符

有时，按上述方式打印美元符是不够的。如果输出结果中的空白有可能被篡改，如支票上的空白，我们是不会想在 $ 与第一个有效数字留有空白的。

例如，如果 Amt 有值为 1.93，输出可能打印为：$　　1.93，在此例中，美元符与第一个有效数字前有一系列空白，有些人会很容易地在中间键入数字，使得 $　　1.93 看去为 $ 99991.93。如果输出结果是支票的话，这将是十分危险的。

为确保美元符紧邻第一个有效数字来打印，可在 LPRINT 或 PRINT USING 语句格式串前用两个美元符，这样可使 $ 浮动，抑制先行逗号和先行零，$ 打印时会挨着数的第一个有效数字。此情形下的 $ 叫作浮动美元符。

考虑下面的例子：

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
28.73	"$$#,###.##"	$ 28.73
4827.26	"$$#,###.##"	$ 4,827.26
927.16	"$$#,###.##"	$ 927.16
0.06	"$$#,###.##"	$ 0.06
87263.27	"$$#,###.##"	$ 87,263.27

§6.3.2.2.6 打印带星号(*)的数字

有时，需要在数前打印先行星号(*)，而不是打印空格或浮动美元符，这叫作星号填充。与浮动美元符相似，星号填充可以最大限度地减小数目被窜改的可能性，例如，支票上的$***15.23比$15.25更难改动.

为得到星号填充，可在LPRINT或PRINT USING语句的格式串开头带两个星号(**)。例如，如Amt有值25.75，则用:

```
LPRINT USING "**#, ###.##"; Amt
```

语句后，结果打印为: *****25.75.

与浮动美元符类似，星号填充也常用于打印支票. 由于左边带带*的数目字与空格相比较难窜改，所以，星号填充能大大降低计算机犯罪的危险性.

总之，输出数值需要防止被改动的程序应当用星号填充或浮动$. 注意到星号与浮动美元符一样，都会替代非有效逗号及先行空白，此外，美元符前用星号会导致同时打印星号和一个浮动美元符.

考虑下面例子:

Amt的值	LPRINT或PRINT USINTG格式串	打印结果
8712.34	"**$#,####.##"	**$8,712.34
27.45	"**$#,####.##"	*****$27.45
283.05	"**$#,####.##"	****$283.05
26.32	"**$#,####.##"	*****$26.32

§6.3.2.2.7 打印加、减号

如果一个变量是负数，则如不另外指定，应在数的左边打印一个减号. 例如，如果变量Amt1的值为—25.59，语名LPRINT USING "####.##"; Amt1会使Amt1打印为—25.59. 然而，我们可能会需要在数的右边打印减号，如财政报表中有时就需要这样。为此，可以LPRINT或PRINT USING语句格式串的右边放一个减号，如下所示:

```
LPRINT USING "###.##_"; Amt1
```

如果变时Amt1的值是正的，则输出中不出现减号，而是空白；如果该值是负的，则其右方打印一个减号.

格式串中用减号会使负数打印时右方带上减号，但如果正数或无符号数，则不打印任何符号。

如想依据变量值的不同，打印加号或减号，可在格式串右边用一个加号，结果如下：

格式串中"+"号的用途

1. 负数后边打印减号
2. 正数和无符号数右边打印加号。

考虑下面的例子，其中示范了格式串中用加号和减号的情形：

Amt 的值	LPRINT 或 PRINT USINTG 格式串	打印结果
−142	"####_"	_142
142	"####_"	142
+142	"####_"	142
−142	"####+"	142_
142	"####+"	142+
+142	"####+"	142+

§6.3.2.2.8 用相同格式串在单行上打印或显示多个编辑值

用户已经见过用一条 LPRINT 或 PRINT USING 语句编排单个变量的例子。多个变量也可用同样的格式串打印。不过，格式串中必须包括空格、分号或逗号来隔开各变量：

```
LPRINT USING "###.## ": Amt1, Amt2
```

格式串中最后一个#后必须跟一个空格以确保各变量不混淆。

类似地，也可用单个格式串来编排数值常量的格式：

例：

```
PRINT USING "###.## ": 012, 105, Amt  ← "###.## "作用于两个常数
                                          012, 105 和变量 Amt
```

§6.3.2.2.9 用下线符打印字符本身

格式串中任何字符前带一个下线符将使字符本身打印出来：

例 子 表	
LPRINT 或 PRINT USINTG 格式串	**打印结果**
PRINT USING "_X##.##"; 12.10	X12.10
PRINT USING "_O##.##"; 12.10	O12.10
PRINT USING "_V##.##"; 2.10	V 2.10
PRINT USING "_C_R##.##"; 12.10	CR12.10

自我测验(二)

1. 为确保在第一个有效字符紧邻处打印一个美元符，应当在格式串开头包括________。
2. 何时用星号填充较为适宜?
3. 写几条语句，把457、38和5输出在三行上，并按最右边的数字对齐.
4. 写几条语句，输入变量叫AnyField，该变量小于1000，四舍五入至三位小数并打印输出。

对于下列情形，指出输出结果

格式串	数据	结果
5. "##.##-"	5.432	
6. "##,###.##"	550.50	
7. "##,###.##-"	550.50	
8. "###.##"	145.432	
9. "###.##"	145.43	
10. "**#,###.##"	375.85	

§6.3.2.3 编辑串变量及串常量

§6.3.2.3.1 利用反斜杠(\)

到目前为止，我们一直在考虑用于编辑和打印数值数据的 LPRINT 和 PRINT USING 语句的各种变型。其实，串数据也可以编辑，在 LPRINT 或 PRINT USING 语句用反斜杠字符(\)可以指定待打印串变量中的字符数。每个反斜杠代表一个字符的位置，反斜杠间的每个空格也占一个字符的位置。考虑下面的 LPRINT USING 语句，其中两

斜杠间有两个空格:

```
PRINT USING "\  \"; "NAME"
```

该语句可使 NAME 一词中的四个字母都打印出来。每个斜杠和每个空格各允许打印单个字符。如果格式串写为"\\"，则只会打印 NA，即每个反斜杠代表一个字符。同样，为打印叫作 B$ 的一个串变量的内容，且至多打印四个字符，可以用:

```
50 PRINT USING "\  \"; B$     (斜杠间有两个空格)
```

LPRINT 或 PRINT USING 产生的字母数字方式输出是左齐的；例如，如果 B$ 的值为 ABC、AB、ABCD，则每执行一次 LPRINT USING 的结果如下:

```
ABC
AB
ABCD
```

考虑下面的例子在这些例子中用了反斜杠字符来编辑串变量:

串变量的数据	LPRINT 或 PRINT USINTG 格式串	打印结果
注意：下面的格式串包括六个字符(4 个空格与两个斜杠)		
ABCDEFG	"\ \"	ABCDEF
ABCD	"\ \"	ABCD
AB	"\ \"	AB

执行以上的 PRINT 或 PRINT USING 语句后，数据打印结果是左齐的。

§ 6.3.2.3.2 "!"格式串

"!"格式串使 LPRINT 或 PRINT USING 语句打印输出串中的一个字符，该字符为串中最左边的那一个。

例:

```
A$ = "X marks the spot"
PRINT USING "! "; A$
```

字母 X 将会打印出来，因为它是 A$ 串中的首字符。

§ 6.3.2.3.3 "&"格式串

如要打印整个串，而不管其内容，可用"&"作为格式串.

例::

```
A$ = "X marks the spot"
PRINT USING "&"; A$
```

整个句子"X marks the spot"将会打印出来，这与 PRINT A$ 功能相同.

注意到格式串的反斜杠、!及&仅用于串变量；如果用于数值变量，将会得到类型失配（type dismatch error）错误．出错情况示例如下:

自我测验(三)

下面语句中有何错误:

1. PRINT USING "##", Amt1.
2. PRINT USING "!"; Amt2.
3. LPRINT "\ \"; "NAME"
4. LPRINT 或PRINT USING 产生的字母数字型输出是________齐的.
5. 用 LPRINT 或 PRINT USING 语句只打印一个字符，格式串中用________.
6. 用LPRINT 或PRINT USING 语句打印整个串，不管其内容，格式串中用________.
7. 两个斜杠间有三个空格的格式串将使________个字符输出.
8. (T、F)格式字符\、! 及&均可用于数值或串变量.
9. (T、F)如果字母内容比格式串允许的间隔长，则字母最右边的若干字符被截去.
10. (T、F)"\ \"格式串将输出与两反斜杠之间空格数目相同的字符.

§6.4 其它打印选择

§6.4.1 使用打印格式表

打印格式表(Printer Spacing Chart)常常用于帮助程序员准备报表。这种表格用于设计报表以使:

(1) 诸标题在整个页面上适当隔开;

(2) 各字母或变量在栏标题下正确对齐，并且在整个页面上均匀隔开。

在本书的例子中，打印格式表分为 80 个打印位置，图 6.4 即是一个样例。单个或多个标题在整个页面上均匀隔开，程序员可以根据表中编号的各个位置确定哪些位置应当放置常数，哪些位置应当保留空白。

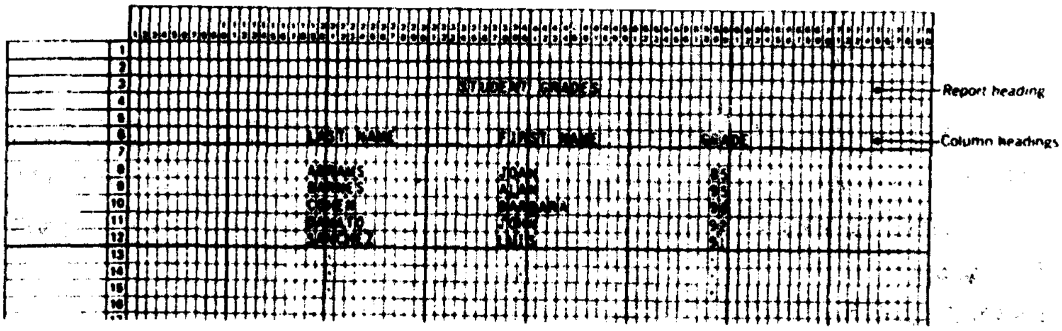


图 6.4 打印格式表样例

§6.4.2 TAB 函数

标题是由串常量或文字组成，放置于输出行或屏幕上的特定位置，用于标识:

(1)报表名;

(2)输出字母名.

含有报表名的行叫作表头(report heading)，含有输出字母名的行叫作栏标题(column heading)。本节将给出生成图 6.4 表中指定的打印输出之程序片段。为本例方便起见，假定每行有 80 个位置。

为使表头中心对齐，需要 STUDENT 一词中的 S 在第 34 个位置上开始，之所以选这一位置，是因为这样可使标题两边空格数相等:

1. 假定输出行包含80个字符.

2. 表头STUDENT GRADES包含14个字符，这也包括STUDENT 和GRADES之间的一个空格。
3. 为使表头放在中央，串常量两边的空格数应当相等，这意味着表头两边应是有 33 个空格：

```
80(每一行位置数) — 14(表头字符数) = 66(剩余空格数)
66／2 = 33(表头两边的空格数)
```

这样，表头应从第34个位置开始。

栏目标题 LAST NAME 、FIRST NAME 及 GRADE 的起始位置也是选出的。这样，这些字母名称可在整个版面上均匀隔开。请再次参见图 6.4 的打印格式表。

TAB 函数与 LPRINT 或 PRINT 语句一起使用可以在打印表格或屏幕上指定位置上输出结果。例如：

考虑下面这条语句：

```
LPRINT TAB(34);"STUDENT GRADES"
```

该语句将使串常量 STUDENT GRADES 在第 34 个位置开始。注意到 TAB 函数与待打印的实际标题或文字之间用分号隔开，且 TAB 与 LPRINT 或 PRINT 语句共用，而不是与 LPRINT 或 PRINT USING 语句用在一起，LPRINT 或 PRINT USING 语句是用于格式编辑的。

PRINT TAB 的格式为：

$$\left\{\begin{array}{l}\textit{LPRINT}\\ \textit{PRINT}\end{array}\right\}\ \text{TAB(n)}$$

n 可以是从 1 到 255 的某一整数，但如果用户的打印机是 80 个字符宽的，则 n 是 1 到 80 的一个数。

TAB 与 PRINT 或 LPRINT 语句共用可以指定输出在打印页面或屏幕上出现的位置。它可以用来对齐一行上的多个录入项，即 TAB 函数可在一条 LPRINT 或 PRINT 语句中出现多次(例如，PRINT TAB(20); "NAME" ; TAB(30); "SALARY"; TAB(40); "DEPT")。但是，如上所述，TAB 函数不能与 USING 选项共用，例如，PRINT USING TAB(20); "##"; Amt 是无效的。总之，TAB 函数常用于 PRINT 或 LPRINT 语句中，用以对齐打印页面或屏幕上标题中的常量或文字。

为在指定打印位置打印已编辑的数值数据，可用两条打印语句，例如：

```
PRINT  TAB(20)   ' This positions the paper at print position 20
' 12.00  will print beginning in position 20
PRINT  USING  "##.##";  12.00
```

PRINT TAB(n)后可以跟一个分号，但如果后边无其它项，是可选的。通常只有当 PRINT TAB(n) 与其它输入项在同一行时才用分号。

假如由于前面的 PRINT 或 LPRINT 指令使屏幕上的光标或打印机的打印机机构超出了 TAB 函数指定的位置，则在此情形下，指定的变量或常量会打印在下一行上，即在 TAB 函数指定的打印位置上。

例:

考虑下面的指令，“LAST NAME”设在位置 40 上，“FIRST NAME”设在位置 30 上：

```
LPRINT  TAB(40); "LAST NAME", TAB(30); "FIRST NAME"
```

输出结果如下:

```
30             40
 |              |
 |              ↓
 ↓             LAST NAME
FIRST NAME
```

下面的程序片段每说明了如何用 TAB 函数得到图 6.4 打印格式表中指定的表头和栏目标题。

```
LPRINT
LPRINT
LPRINT TAB(34); "STUDENT GRADES"
LPRINT
LPRINT
LPRINT TAB(19); "LAST NAME"; TAB(38); "FIRST NAME"; TAB(58); "GRADE"
LPRINT
LPRINT TAB(19); LastName$; TAB(38); FirstName$; TAB(59); Grade
END
```

在最后一条 LPRINT 语句中有 TAB，这样各变量内容在打印时直接在相应栏标题下对齐。例如变量 Grade 的 TAB 设在位置 59，这意味着该变量的内容将会在栏标题“GRADE”下中心对齐，“GRADE”一词打印在第 58～62 个位置。

§ 6.4.3 Width 语句

Turbo Basic 假定用 LPRINT 语句打印或用 PRINT 语句显示的每一行包含 80 个字符，这即意味着依据缺省设置，打印或显示了 80 个字符后会输出一个回车符，这样纸或屏幕显示会向前进一行。

不过，许多打印机每行可打印 100、120 或 132 个字符，类似地，屏幕每行可显示 40 或 80 字符。

打印机或屏幕的 80 个字符的缺省宽度都可以按下述格式改变:

```
    WIDTH [ "SCAN:" ], size
或:
    WIDTH [ "LPT1:" ], size
```

将显示宽度改为40个字符，可用:

```
WIDTH "SCAN:", 40
```

或:

```
WIDTH "SCAN:", 40
```

设备名"SCRN"是缺省设备，这样，如是要改变显示宽度，"SCRN"可以完全从语句中省去。

如要改变一个打印行的宽度，可用

```
WIDTH "LPT1:", n
```

其中: n可以为0～255间的某一整数。常用于打印行的WIDTH语句如下:

WIDTH语句示例

```
WIDTH "LPT1:", 100
WIDTH "LPT1:", 120
WIDTH "LPT1:", 132
```

为确保正常运行，WIDTH"LPT1"语句必须在所有LPRINT语句前，相应地，WIDTH "SCRN:"必须在所有PRINT语句前。

如果用户的计算机配有多台打印机，则"LPT2"与"LPT3"也是有效的设备名。

总之，WIDTH语句决定了一页或一屏上每一行可打印或显示的字符数。如果省略，则假定为80个字符宽。

§6.4.4 SPC函数

SPC(n)指令可在一行上跳过n个空格，例如，为在Amount1和Amount2之间留出10个空格，可以用

```
LPRINT  Amount1; SPC(10); Amount2
PRINT   Amount1; SPC(10); Amount2
```

注意到SPC函数与TAB函数一样，只能与LPRINT或PRINT语句一起使用，而

不能与 LPRINT 或 PRINT USING 语句共用. LPRINT 或 PRINT USING 用于编辑输出结果.

§ 6.4.5 控制一行的打印

§ 6.4.5.1 在一条 PRINT 或 LPRINT 语句后用一个分号

假如我们希望在单独一行上打印一系列变量, 每个变量都分别录入或计算, 可以在 PRINT 或 LPRINT 指令后用一个分号, 这样打印的即可把数据放在一行上:

例:

输入 10 个变量, 并打印在一行上.

```
FOR I = 1 TO 10
        INPUT Amount
        LPRINT AMOUNT; ' 注意: LPRINT 语句的末尾为" ; "
NEXT I
LPRINT
END
```

LPRINT Amount; 指令末端的分号将使所有 10 个变量放在单独一行上, 但是, 在未到达行结束处时, 这一行数据是不会打印出来的, 即如果 10 个变量用了所有 80 个打印位置, 则整行将会输出, 否则, 需要单另一条 LPRINT 指令来强制回车. 这样在整个页面已定位的 10 个变量才会打印出来. 上例中跟在 NEXT 语句中的 LPRINT 会强制打印 10 个变量.

这样, 一条 LPRINT 指令末尾的分号可用于在整个页面上数据的定位, 但只有在两种情况下才会打印: (1)输出定位时到达一行的结束处; (2)另一条 LPRINT 强制使前一行打印.

注意到上例中, 用 PRINT 代替 LPRINT, 结果是不同的. 因为 INPUT 语句会使前面的输出显示在屏幕上, 这样, 每个值将会显示在单另行上.

§ 6.4.5.2 在一条 LPRINT 或 PRINT 语句后用一个逗号

LPRINT 或 PRINT 指令后也可用逗号来给数据定位:

```
FOR I = 1 TO 10
        INPUT Amount
        LPRINT AMOUNT,   注意: LPRINT 语句的末尾为" , "
NEXT I
LPRINT
```

```
END
```

用户可能还记得逗号把一行分成一系列区域．大多数 PC 机有五个区域，前四个区域最多可以打印或显示 14 个字符，而最后一个区域最多可以显示 24 个字符．假定有五个区域，且一条 LPRINT 或 PRINT USING 语句末端有一个分号(如，LPRINT Amount，)则当五个值输出至屏幕或打印机时，计算机会显示或打印一行．因此，上述程序片段会产生如下形式的输出：

```
123    487    211    672    783
441    113    222    221    655
```

即打印出 10 个变量值，第一行五个，第二行五个．

如果用了 LPRINT Amount，语句，则当有五个数据在打印机上定位后，打印才会开始．

假如编写下面这段程序：

```
FOR I = 1 TO 10
        INPUT  Amount
        LPRINT  AMOUNT, ' 注意此处的“，”
NEXT I    ' 打印第二行
LPRINT
END
```

上述程序段中的 FOR…NEXT 导致五个数值在一行上输出，而只有两个数值的第二行打印必须强制执行，因为该行的另三个区域尚未填充．LPRINT Amout，语句只有当五个区域都填充后才开始打印．因此，只有在 FOR….NEXT 循环后用一条 LPRINT 语句强行使第二行打印输出．

在本例中，如果省略了第二条 LPRINT 语句，则只能输出前五个值．为了使后两个值打印出来必须第二 LPRINT 语句强制打印．

§6.4.5.3 PRINT、LPRINT TAB(n)或 SPC(n)与 PRINT 或 LPRINT USING 联合用于数据的定位和编辑

为将各数据在整个页面上定位，可用带 TAB 或 SPC 函数(或二者)的 PRINT 或 LPRINT 指令，另外，在下一行用 PRINT 或 LPRINT USING 产生编辑后的输出结果．

考虑图 6.5 中的打印格式表：

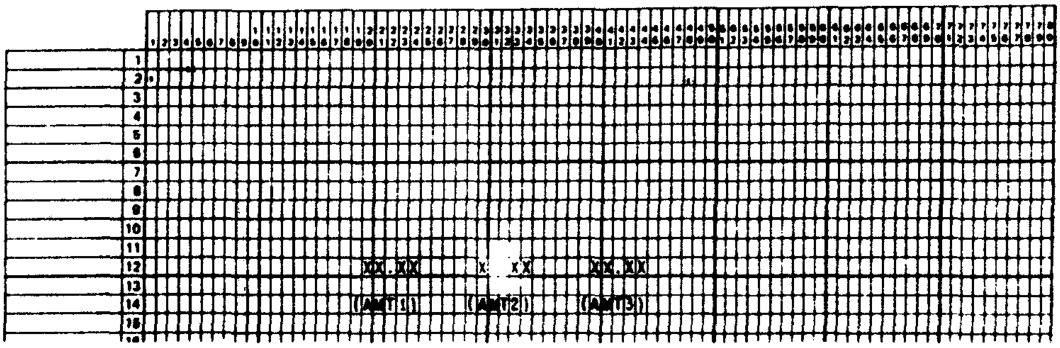


图 6.5　打印格式表样例

在打印表指定的精确位置上打印已编辑数据的一种方法是用 TAB 或 SPC 函数:

```
LPRINT  TAB(20);
LPRINT  USING  "##.##"; Amt1;
LPRINT  SPC(5);  ' PRINT  TAB(30)可替代此语句
LPRINT  USING  "##.##"; Amt2;
LPRINT  TAB(40)LPRINT  USING  "##.##"; Amt3;
LPRINT             ' 输出一行
```

§ 6.4.6　进纸至新页

§ 6.4.6.1　用 CHR $ (12)进纸

计算机生成的报表常常在连续表格纸上打印．连续表格纸各页相连，每页末端有穿孔，用于把各页分开．生成报表后，这些表格纸或页面必须分离开来(参见图 6.6 所示的连续表格纸).

一般情况下，计算机写一行，纸进一行，最终从一张纸打至下一张纸，而不管每一张是单独一页这个事实,．有时还可能打在穿孔上，造成阅读困难．

尽管打印时计算机自动逐页打印连续表格，但编程时应使计算机识别页的外型．这样，每页开头带有报表及栏标题，随后是作为报表主要成分的数据行，当到达一张纸的底部时，跳至新页并在其上再次写上标题．

ASCII 码 12 是计算机的换页符，用它可使纸走至下一页顶部．为使纸走至下一页顶部，可以用 CHR $ 函数与换页符的 ASCII 码，即 LPRINT CHR $ (12)，这将使纸走到新页顶部:

```
LPRINT  CHR$(12)     ' CHR$(12)为换页符的 ASCII 码
```

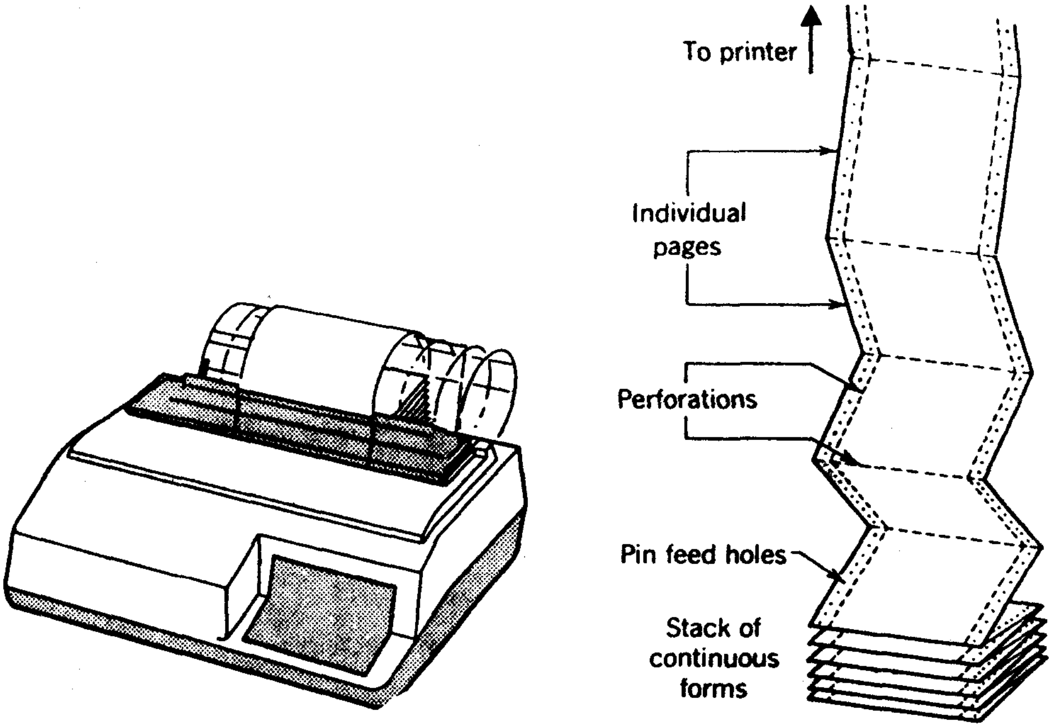


图 6.6 连续表格纸样例

下面是另一种将纸走至下一页顶部的方法；该方法较为明晰，本书建议用户采用这种方法：

```
NewPage$  =  CHR$(12)
LPRINT   NewPage$
```

因此，当需要报表进纸至下一页顶部时，可用 LPRINT CHR$(12)或上面推荐的方法．我们还可以将换页符与 TAB 函数结合起来使用，如下所示：

```
LPRINT  CHR$(12); TAB(34); "STUDENT  GRADE"
```

§6.4.6.2 用编程的行计数器来控制页的结束

为确保不在连续表格纸的穿孔处打印信息，需要测试是否到达一页的结束处．测试是否到达一页结束处的最佳方法是用一个编程的行计数器．用这样的计数器控制页的结束，必须按下述方式进行：

编程的行计数器

1. 建立一个行计数器变量，初始化为零.
2. 确定每页想要打印的行数.
3. 每一次执行LPRINT 语句后，将行计数器变量增1,这样，行计数器变量的数总是等于实际打印的行数.
4. 每一次执行LPRINT 语句后，测试行计数器变量，看其是否等于每页的期望打印行数.

   a. 如果变量等于期望值，用一条LPRINT CHR $(12)语句将纸进至下一页顶部，在该页顶部打印标题，然后把行计数器重新初始化为零.
   b. 如果行计数器的值小于期望值，则继续执行程序.

需要打印若干页时,程序中应当用编程的行计数器. 否则，打印机只能一行接一行打印，而不管页的轮廓或穿孔处.

下面将要用到的一个编程的行计数器按照每打印一行增 1 的方式进行，写完指定数目行后(如 50)，可以简单地指令计算机在新页上打印标题.

例 1: 逐行打印计数

```
LineCtr = 0
    .
    .
    .
    LPRINT ...                  'Prints data line
    LineCtr = LineCtr + 1
    IF  LineCtr = 50 THEN
        GOSUB PrintHeadings     'Headings are printed in a subroutine
    END IF
    .
    .
    .
END
PrintHeadings:
    LPRINT CHR$(12)             'Advance to top of new page
    .
    .
    .                           'Print Headings here
    LineCtr = 0                 'Reinitialize Line Counter
RETURN
```

注意，每一次执行 PrintHeadings 子程序时，LineCtr 应当重新初始化为零. 下面程序中用了同样的方法，但是用的是隔行打印.

例 2: 隔行打印计数

```
LineCtr = 0
LPRINT ...                  ' 打印数据行
LPRINT                      ' 打印空行
LineCtr = LineCtr + 1
IF LineCtr = 25 THEN   GOSUB PrintHeading
END IF
```

上述两例都假定每一页有 50 行，在第一个例子中，实际打印出 50 行数据，而在第二个例子中，打印出的是 25 对双间隔行，即 25 行数据及 25 行空白。

自我测验(四)

1. 测试表格纸上最后打印行通常用一个________.
2. 编写一组指令打印一条记录并空两行，打印完30行后，执行PrintHcadings(打印标题)子程序.
3. 报表用于辅助制作报表.
4. 编写一组指令打印中心对齐的表头"INVENTORY RBPORT"，空两行，然后在页面上打印均匀隔开的栏标题"ITEM NUMBER"、"DESCRIPTION"、"ON HAND"与"COST".
5. (T或F)TAB函数可与LPRINT、LPRINT USING、PRINT或PRINT USING语句用在一起.
6. (T或F)如果光标已超过TAB函数中指定的位置，则光标将退回至所在行正确的位置上.
7. ________语句可用于改变一个打印行或屏幕显示行上允许的字符数.
8. 写一段解答第4题的程序，页宽设为120个字符宽并且栏标题之间留14个空格.
9. 把纸进至下一页的顶部要用________。
10. 编写一组指令打印下列栏标题:

```
Heading line 1     Column 7     EMPLOYEE
                   Column 30    HOME
                   Column 45    HOME
Heading line 2     Column 9     NAME
                   Column 28    ADDRESS
                   Column 45    PHONE
```

§6.4.7 打印页号

打印标题时，常常需要页号作为表头的一部分。考虑图 6.7 打印格式表上的表头行:

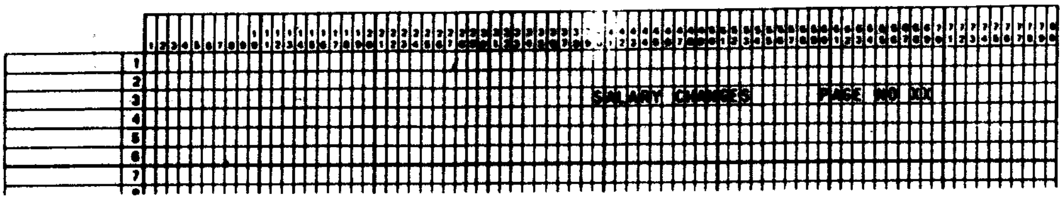


图 6.7　关于表头行的打印格式表

在表头行上打印页号的指令组应包括:

```
PageNo = 1
GOSUB PrintHeadings
  .
  .
  .
END
PrintHeadings:
    LPRINT CHR$(12)
    LPRINT TAB(40); "SALARY CHANGES"; TAB(60); "PAGE NO"; TAB(68); PageNo
    PageNo = PageNo + 1
    LineCtr = 0
RETURN
```

实际计页数时要建立一个初始值为 1 的数值变量 PageNo. 每次执行 Print Headings 子程序时, 页码计数器 PageNo 打印出来, 然后增 1.

§ 6.4.8　打印运行日期

表头由报表名组成, 对于多页报表而言, 还包括页号, 在大多数情况下, 也包括运行日期.

打印运行日期的一种方法是按需要的格式读入实际日期, 这个日期随后移至报表标题内.

但是这种获得日期的方法是低效的, 因为这需要:

(1) 读入日期的附加输入指令;
(2) 按正确格式输入日期还需另外击键.

如果用户在 DOS 下输入日期, 或者系统有一内设日历/时钟, 则可直接从系统得到日期, 而无需另外输入数据.

用 DATE $ 函数可以取得系统日期, 两种形式的系统日期皆可:

(1) 开机时输入的;
(2) 系统日历/时钟自动存储的. 该日期可以传递至用户程序的一个串变量中.

```
[LET]  字符串变量 = DATE$
```

例:

```
DateOfRun$ = DATE$
LPRINT TAB(40); "Date is: "; DateOfRun$
```

或:

```
LPRINT TAB(40); "Date is: "; DateOfRun$
```

DATE$是一个长度为10的字符串，格式为mm—dd—yyyy．mm表示月份，dd表示日期，而yyyy表示年份(如12—25—1990)．如不用在操作系统下输入的日期，可用下面这种方式:

```
PRINT "ENTER DATE as mm-dd-yy"
INPUT DateIn$
INPUT Date is: "; DateIn$
```

输出结果

```
Date is: 02-15-91
```

如果在操作系统下输入时间或者系统有内设日历／时钟，用户也可以从系统取到时间，如要将时间存在一个串变量中，可用下列格式:

```
[LET]  字符串变量 = TIME$
```

例:

```
TimeOfRun$ = TIME$
LPRINT TAB(60); TimeOfRun$
```

或:

```
LPRINT TAB(60); TIME$
```

TIME$为一个长度为8的字符串，格式为hh:mm:ss；其中：hh表示小时(24小时制),mm表示分钟，而ss表示秒．

设计报表的编程准则

1. 在每页顶部应包括一个标识报表的表头.
2. 在表头中应包括日期和页号.
3. 应包括若干栏标题，用于标识待打印的每一字段或变量.
4. 将最重要的变量或字母放在左边，使其最引人注目.
5. 用LPRINT USING 或PRINT USING语句编辑数值字段，增强易读性.
6. 在报表末端，或者如果合适的话，在每一页末端，应包括总计.
7. 用 * 号标识总计的级别.

例:

```
DEPT TOTAL IS $33,266.25*      ← 低级总和(一个*)
        :
        :
        FINAL TOTAL IS $167,,267.53**    ← 高级总和(两个*)
```

8. 用多张纸时，应测试报表中页结束条件.
9. 用一句类似"End of Report"的话结束报表.

§6.5 打印不种类型的报表

用输入数据文件制作的打印报表分为三类:

§6.5.1 明细或事务报表

这一类报表对文件中的每条记录都相应有特定的输出. 从主应收账文件生成的客户账单就是明细报表的一个例子，每条输入记录都可生成一行或多行输出. 类似地，从主工资单文件生成的工资单支票也是明细报表. 再有，一个公司发送的零件号清单也是明细报表. 用明细报表提供个性化的输入的是最为有效的,此类输出如账单或支票. 由于明细表的每条读入的记录都要产生输出,所以很耗时，例如，从一个有 300,000 条记录的文件打印 300,000 行可能需要几个小时. 另外，此类报表提供的是最详尽的数据，可能并不适合于各种应用，当需要总计或一览表时，尤其如此.

§6.5.2 例外表

有时用户要求打印的明细表，但事实上，其它类型的输出可能更为适宜。例如，假定一个保险代理人要求一份所有客户的清单，其中要列出这些客户最后一次付款的日期。当问到为何需要明细表时，代理人回答说需要用此清单与过去 90 天内未付款的所有客户联系。一位有经验的计算机专业人员会建议另外一种输出，只列出满足保险代理人所设标准的那些客户，即列出 90 天内未付款的那些客户，这样可以节约打印所有投保人要花费的时间和精力，但更重要的是，可以使保险代理人的工作变得容易。代理人可得到一份需要联系的那些人的清单，而无需从大量明细行中筛选。

这类输出叫作例外表。它是满足(或不满足)某些准则的个别记录的清单。例外表的其它例子有:

65 岁以上的雇员清单，仓库中现有数量低于某一最小值的那些零件号的清单。

§6.5.3 一览表

顾名思义，一览表是总结性的，而不是逐项列出。如果一览表或总计比明细表或例外表可为用户提供更为综合的输出，则应生成此类报表以作为详细输出的后选报表。

通常，如果报表是为了用户服务的，计算机专业人员则应考虑生成例外表或一览表，而不是明细表。生成明细表需要花费较长的时间，且往往体积庞大，常常需要进行一些总结才会对用户或经理们有意义。

§6.5.4 明细表、例外表及一览表的示范打印程序

考虑下面的输出文件，它包括雇员名、社会保险号、雇佣部分及周薪:

```
"C. White",476329582,1,650,00
"J. Jones",578462193,1,875.00
"K. Smith",,2,150.00
"L. Love",538461422,2,525.00
,4893934763,3,450.00
"M. Meyers",515967343,3,725.00
"T. Brown",4567784742,0,975.00
```

§6.5.4.1 明细表示例

该明细表列出了所有雇员，它们的社会保险号，各自的雇佣部门及周薪。该报表的打印格式表如图 6.8 所示。

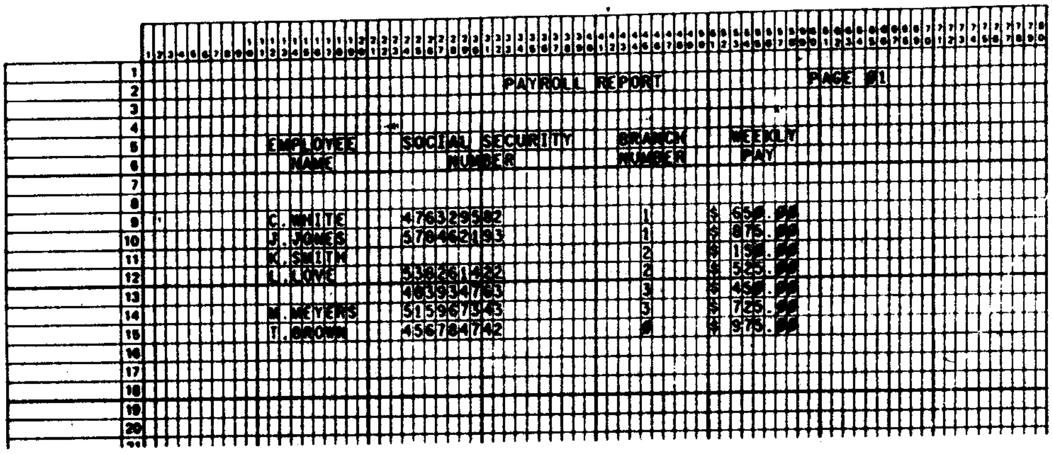


图 6.8　明细报表的打印格式表

§ 6.5.4.2　例外表示例

该例外表列出满足下述条件的雇员：

a. 漏掉雇员名.
b. 漏掉社会保险号.
c. 部门号不是1，2或3.
d. 周薪不在 251.00 及 1,000 美元之间.

有些输入记录漏了数据，漏掉一个字母用连续键入两个逗号表示，如第三条记录中的情形，或者，如果第一个定母漏掉了，则用一个先行逗号，如第五条记录中的情形. 这个例外报表的格式如图 6.9 所示.

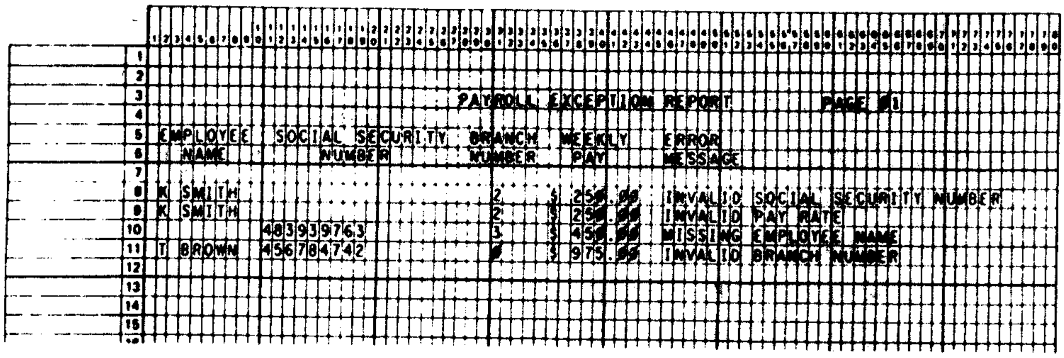


图 6.9　例外报表格式

§6.5.4.3　一览表示例

该报表中打印出了各部门的总工资，其格式如图 6.10 所示。

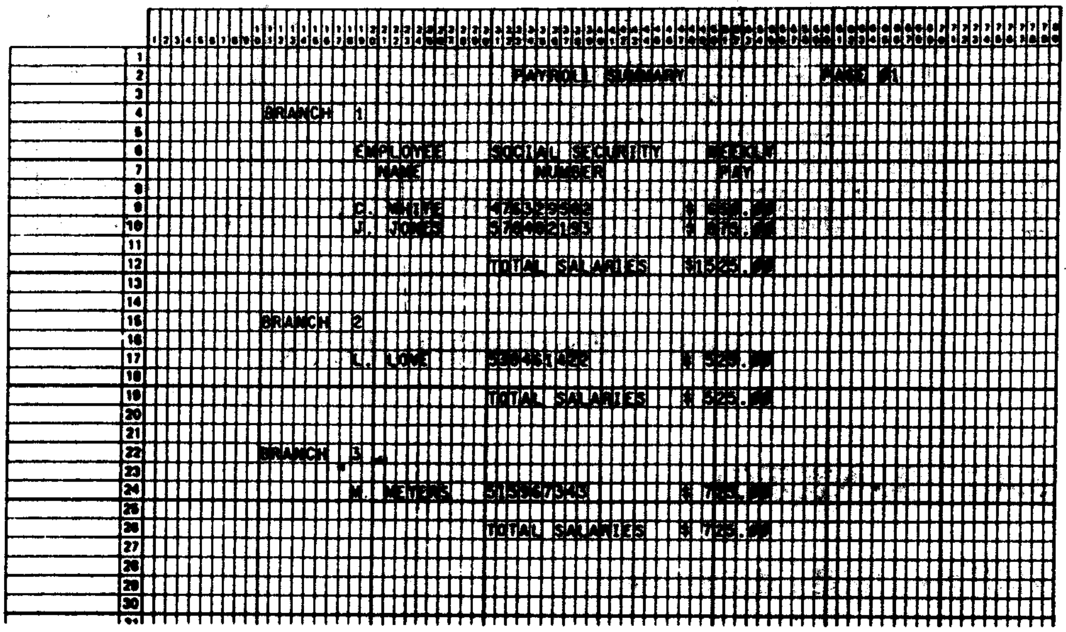


图 6.10　一览表格式

§6.6　本章辅导

§6.6.1　本章小结

I　文件处理导论

A．用交互处理方式，从终端接受输入并立即产生输出。
B．用批处理方式，记录从文件读入，所在记录一次即可处理完毕。
C．文件可以是顺序、随机或二进制的。
D．所在文件都包含记录。
　1．记录为文件中的一个信息单位。
　2．每条记录由一外或多个字段组成。
　3．字段为记录中的数据项。
　4．每个字段由一个或多个字符组成。
E．建立文件时，验证输入数据非常重要。
F．顺序文件必须总是按顺序处理。

1. 建立文件前，必须先用OPEN语句打开.
2. 往一顺序文件输出数据，用WRITE语句.
3. 每个文件应当用CLOSE语句关闭.
4. 从文件读入记录，用INPUT#语句.
5. 读完每条记录后，应检测文件结束(EOF)与否.

Ⅱ 打印输出的特点

A. 显示输出最常用于回答查询、提供交互式输出.
B. 打印报表是用户保存使用的正式文档.
C. 输出必须清晰、整洁并便于阅读.
D. 打印和显示输出所用方法:
  1. 打印数据必须进行编辑.
  2. 表格必须适当间隔.
  3. 标题必须打印.
  4. 数据必须定位.

Ⅲ 编辑功能

A. 编辑用PRINT USING 和LPRINT USING 语句完成.
B. 数值变量或字母的编辑可用下列方法完成:
  1. 格式串中用#号可以把数值数据右对齐并在左边填充空白.
  2. 在格式串中插入一个小数点可以对齐有小数位的字段.
  3. 在格式串中插入美元符或逗号，便于在需要的地方打印出来.
    a. 单个美元符会使所有美元符对齐.
    b. 两个美元符会使美元符"浮动"，这样可在第一个有效数字紧前方打印美元符.
  4. 在格式串中用两个星号可以使核对保护符"*"替代所有先行空格.
  5. 在数的左、右边用加、减号可使符号打印出来.
    a. 如果值为负，则减号会打印出.
    b. 加号会使正值打印时带加号，而负值带减号.
C. 串变量或字段的编辑可用下列方法完成:
  1. 在格式串中用两上反斜杠间带空格的方法指明待打印字符的准确数目.
  2. 在格式串在用"!"打印变量或字段的首字符.
  3. 在格式串在用"&"打印整个变量或字段.

Ⅳ 其它打印选项

A. 打印格式表用于辅助设计打印输出.

B. TAB函数用于指明一个条目出现的确切列号.

1. TAB函数常用于报表和栏目标题的放置.

2. TAB函数与PRINT或LPRNT语句共用，但不能与PRINT USING或LPRINT USING 语句共用.

C. WIDTH语句可用于调节每一行上打印位置的数目.

D. SPC函数用于在一行上留出指定数量的空格.

E. 行打印的控制.

1. PRINT或LPRINT 语句后的一分号允许若干数据显示或打印在一行上.

2. PRINT 或LPRINT 语句后的一个逗号会使变量定位于若干区域内，前四个区域各有 14 个位置，最后一个区域有 24 个位置.

3. 当前一条PRINT 或LPRINT 语句的最后一个字符是分号或逗号，常常需要另一条 PRINT 或 LPRINT，它将强制显示或打印.

F. 走纸至新页:

1. CHR $ (12)强制换页.

2. 一个编程的行计数器可用于确定何时走纸至新页.

G. 一个编程的页码计数器当用于在每页上打印页号.

H. 日期可用变量DATE $ 从计算机获得，时间可用变量TIME $ 得到.

I. 打印不同类型的报表:

1. 明细报表用于给每一输入生成特定输出.

2. 例外报表列出满足(或不满足)某些准则的单个记录.

3. 一览表为若干组输入生成输出，是总结性的.

§6.6.2 本章自测

在第 1～10 题中，在缺了结果的一栏中填入合适的值:

	格式串	变量内容	编辑结果
1.	###.##	000.05	
2.	###.##	001.05	
3.	$#####.##	0026.54	
4.	###—	002	
5.	##—	4	
6.	$$##.##	000.00	
7.	$$##.##	025.00	
8.	###	008	
9.	#####	00123	
10.	#,###	0002	

11. 打印报表上会出现的两类标题是________和________.

12. ________是用于待打印数据定位的一种工具.

13. 跳至新页，可以用________.

14. 得到运得日期并打印可以用________.

15. ________语句用于指定一行上位置的数目.

16. 如果对于每一输入都需要特定的输出，应当生成________报表.

17. 如果经理需要一份存货中需再次订购的项目清单，需要________报表.

18. 如果经理想看商店(不是商店内的某个部门)的周销售数据，需要________报表.

19. 用TAB函数写一条语句使报表标题"INVENTORY REPORT"显示在第32列上.

20. 写一条语把37750.25编辑为 $ 37,750.25.

21. (T或F)我们通常把PRINT TAB….与PRINT USING 指令放在单另行上.

22. (T或F)用单独一条PRINT语句编辑整个一行是可能的.

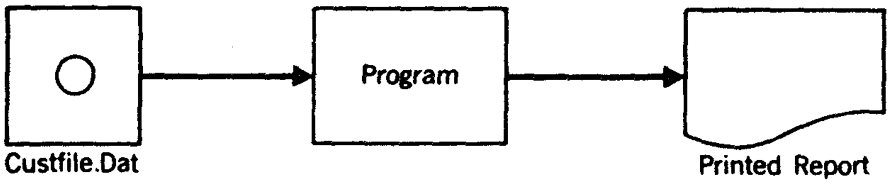
23. (T或F)如果待编辑字段小数点右边的位数大于格式串中指定的数，则多余的数字不打印.

24. (T或F) 如果待编辑字段小数点左边的整数位大于格式串中指定的数，则多余的数字将被四舍五入，且看不出有效数字已被截去.

25. (T或F)用TAB函数在一行上退格是可能的.

§ 6.6.3 实例

考虑下面问题描述:



Custfile.Dat 记录格式

CustNo, CustName$, Amt

打印报表

打印报表由程序员设计

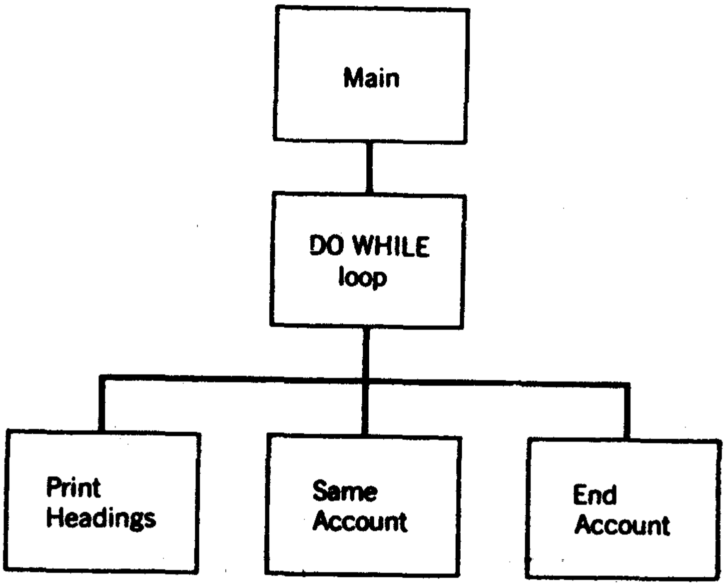
写一个包括下述内容的程序:

1. 在每页上打印标题.

2. 在每页上打印页号。
3. 明细行间应空出一行。
4. 打印运行日期。
5. 在每页上打印新标题。

该程序打印文件中的每一条记录，并且打印出每 5 个记录数量字段的总计。注意：每 5 个记录的 Cust Name $ 一样，磁盘记录的数目是 5 的整数倍。图 6.11 包括伪代码及层次图，图 6.12 所示的 Turbo Basic 程序。

层次图



伪代码

```
START
    清屏
    打开输入文件
    初始化页号
    换页
    DO WHILE 还有输入?
        PREFOR
            打印标题
        ENDPERFORM
        DO 处理五次
            读入一条输入记录
            PREFOR
```

```
                加入总数
                打印该记录
            ENDPERFORM
        ENDDO
        PREFOR
            打印总数
        ENDPERFORM
    ENDDO
STOP
```

图 6.11　实例伪代码及层次图

```
CLS
OPEN "Custfile.Dat" FOR INPUT AS #1
Total = 0
PageNo = 0
NewPage$ = CHR$ (12)
DO WHILE NOT EOF(1)
        GOSUB PrintHeadings
        FOR I% = 1 TO %
                INPUT #1, CustNo, CustName$, Amt
                GOSUB SameAccount
        NEXT I%
        GOSUB EndAccount
LOOP
CLOSE #1
END

PrintHeadings:
        PageNo = PageNo + 1
        LPRINT NewPage$
        LPRINT TAB(7); "PAGE ";
        LPRINT TAB(15);
        LPRINT USING "##"; PageNo;
        LPRINT TAB(45); DATE$
        LPRINT
        LPRINT TAB(7); "CUSTOMER"; TAB(23); "CUSTOMER";
```

```
        LPRINT TAB(33); "TRANSACTION"
        LPRINT TAB(8); "NUMBER"; TAB(25); "NAME";
        LPRINT TAB(35); "AMOUNT"
        LPRINT
        LPRINT
RETURN

SameAccount:
        Total = Total + 1
        LPRINT TAB(10); CustNo;
        LPRINT TAB(23); CustName$
        LPRINT TAB(35)
        LPRINT USING "$ $##.##"; Amt
        LPRINT
RETURN
EndAccount:
        LPRINT TAB(10); "Total due: "
        LPRINT TAB(25)
        LPRINT USING "$ $,####.##"; Total
        Total = 0
RETURN
```

图 6.12(a) 实例的程序

```
12,"Stern",34.45000076293945
12,"Stern",45.56999969482422
12,"Stern",37.95999908447266
12,"Stern",43.56000137329102
12,"Stern",51.5
47,"Meyers",17.75
47,"Meyers",20.95000076293945
47,"Meyers",32.27000045776367
47,"Meyers",44.90000152587891
47,"Meyers",50.9900016784668
123,"Jones",19.95000076293945
123,"Jones",22.70000076293945
123,"Jones",33.40000152587891
123,"Jones",40.88000106811523
123,"Jones",56.77999877929688
135,"Smith",15.57999992370605
135,"Smith",22.89999961853027
135,"Smith",31.45000076293945
135,"Smith",44.22000122070312
135,"Smith",51.5
```

图 6.12(b) 实例的输入样例

PAGE 1 02-15-1990

CUSTOMER NUMBER	CUSTOMER NAME	TRANSACTION AMOUNT
12	Stern	$34.45
12	Stern	$45.57
12	Stern	$37.96
12	Stern	$43.56
12	Stern	$51.50

Total due: $213.04

PAGE 2 02-15-1990

CUSTOMER NUMBER	CUSTOMER NAME	TRANSACTION AMOUNT
47	Meyers	$17.75
47	Meyers	$20.95
47	Meyers	$32.27
47	Meyers	$44.90
47	Meyers	$50.99

Total due: $166.86

PAGE 3 02-15-1990

CUSTOMER NUMBER	CUSTOMER NAME	TRANSACTION AMOUNT
123	Jones	$19.95
123	Jones	$22.70
123	Jones	$33.40
123	Jones	$40.88
123	Jones	$56.78

Total due: $173.71

PAGE 4 02-15-1990

CUSTOMER NUMBER	CUSTOMER NAME	TRANSACTION AMOUNT
135	Smith	$15.58
135	Smith	$22.90
135	Smith	$31.45
135	Smith	$44.22
135	Smith	$51.50

Total due: $165.65

图 6.12(c) 实例的输出样例

§ 6.6.4 复习题

I 是非题

1. 当输出结果打印时，常要进行编辑.
2. 单个格式串可用于打印一个或多个变量.
3. 如果一个变量的值没有符号，则决不会打印为负值，既使格式串中有负号也不会.
4. 如不用格式串，11.55可能打印为1155.
5. 加、减号常常作为最右边的字符传递给计算机.
6. 下列语句有效:

```
PRINT USING TAGB(20); "##"; Amt
```

7. 在到达当前页结束处理时，用CHR $ (12)进纸至新页顶部.
8. 用SPC函数在一行上退格是可能的.
9. 存取顺序文件，下列语句是效的.

```
OPEN "O", #1, "EMPLOYEE.DAT"
```

10. 将数据输出至一个顺序文件，下列语句是有效的.

```
PRINT #1, EmployeeName, SocSecNo, PayRate
```

11. 从顺序文件读人信息用INPUT语句.
12. 必须打开和关闭所有顺序文件.
13. 在把各字母写至顺序文件之前应当进行适当的编辑.
14. 数据确证在数据要写至文件时尤其重要的.
15. 结尾值用于检测文件结束.
16. 计算机生成的显示和报表应当把简明和高效置于用户易读性之上.
17. 存储在文件上的数据应当把简明和高效置于用户易读性之上.
18. 打印格式表用于告诉计算机输出显示出来的是什么样子.
19. 编辑的目的是使数据更清晰、更简洁、更便于使用.
20. 格式串"#"用于左对齐数值数据.
21. LPRINT 语句用于在屏幕上显示信息.
22. 当数值字段中的数值被截去后，会出现长度错误.
23. 用双精度数可增强一个数值变量或字母的精度.
24. 在格式串中，包括与小数点前位数相同数目的美元符，可以使美元符"浮动".

Ⅱ 综合题

在第 1～15 题中，填入缺少的项.

	变量内容	输出结果 格式串	内容
1.	754.50	"###.##"	
2.	754.60	"$#,###.##"	
3.	35.64	"$ $#,###.##"	
4.	George Washington	" !"	
5.	Charles Babbage	"&"	
6.	Abraham Lincoln	"\ \"	
7.	4536.27	"$ * * , * * * ,##"	
8.	−343.23	"###.##_"	
9.	−34743.46	"#,###.##−"	
10.	−5647.38	"#,###.##+"	
11.	345.62	"_C_R$###.##"	
12.	3.3	"_V#.#"	
13.	3750500.00	"$###########,.##"	
14.	3750500.00	"$#,###,###.##"	
15.	−375.50	"###.##+"	

8-320 B20

§ 6.6.5 调试练习

改正下列程序中的错误:

1.

```
CLS
LET Total = 575.95
PRINT USING "$**.**"
END
```

2.

```
CLS
LET A$ = "PRICE"
LET B$ = "QUANTITY"
LET C$ = "TOTAL"
LPRINT USING "\    \"; SPC(5); "\        \"; A$,B$,C$
END
```

3.

```
CLS
OPEN #1, "O", "SALES.DAT"
READ SalespersonNo
PRINT #1, SalespersonNo
FOR Counter = 1 TO SalespersonNo
     READ Branch$, Amount
     PRINT #1, Branch$; Amount
NEXT Counter
DATA 5
DATA "Main", 4500, "Sub1", 4800, "Sub2", "5200
DATA "Sub3", 4750, "Sub4", 5475, "Sub5", 5800
END
```

4.

```
CLS
WIDTH 30
PRINT TAB(8); DATE$; SPC(6); "MONTHLY PAYROLL REPORT";
PRINT SPC(10); PageNO
END
```

§ 6.6.6　编程题

每道编程题应包括以下内容:

a. 在每页上打印标题.
b. 在每页上打印页号，作为表头的一部分.
c. 在每页上打印运行日期，作为表头的一部分.
d. 包括一个页结束子程序走纸至新页顶部，并且打印完25行的细项后，打印适当的标题信息.
e. 明细行隔行打印.

注意到为测试一个程序(业已假定输入文件存在的程序)，用户可以另写一段程序生成输入文件，或者用正文编辑器来生成该文件.

1. 明细表

基于图 6.13 所示问题描述，编写一个程序.

注意:

a. 此问题用到下面这些公式:

总　量 = 数量×单价
贴现量 = 总量×贴现百分比
净　量 = 总量—贴现量

所有结果应四舍五入到美分.

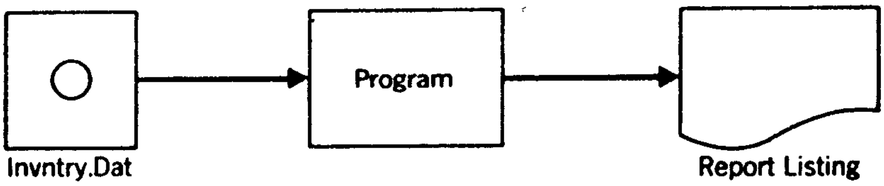
b. 消去单价(Unit Price)、数量(Quantity)、总量(Gross Amount)、贴现量(Discount Amount)及净量(Net Amount)中无用的零.

c. 在负值字段右边打印减号，输入记录中的负值数量(Quantity)字段表示贷

20A8-32V

方项目收入.

系统流程图



Invntry.Dat 记录格式

ProdNo, ProdDescription$, WarehouseNo, Qty, TerrCode, UnitPrice, DisctPct, CustNo

(对贷款方而言，Qty 可能为负)

报表打印格式图

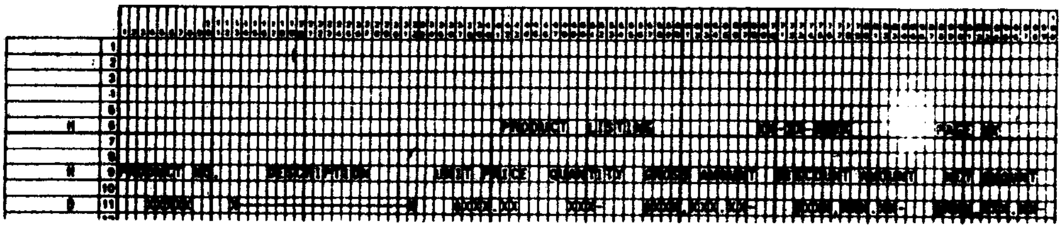


图 6.13　编程作业 1 的问题描述

2. 明细表

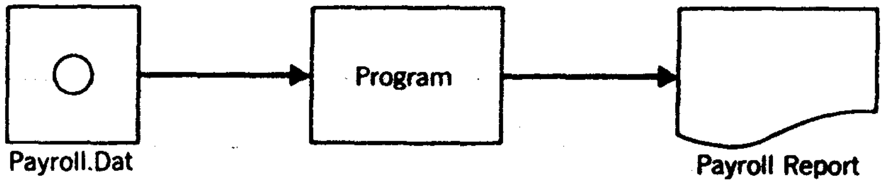
基于图 6.14 所示问题描述，编写一个程序.

注意:

a. 工资前要用浮动美元符，并按需要打印逗号.

b. 消去未休病假，未休假期及报表末行总计中无用的零.

系统流程图



Payroll.Dat 记录格式

EmpNo, EmpName$, Salary, UnusedSickDays, UnusedVacationDays

工资单打印格式图

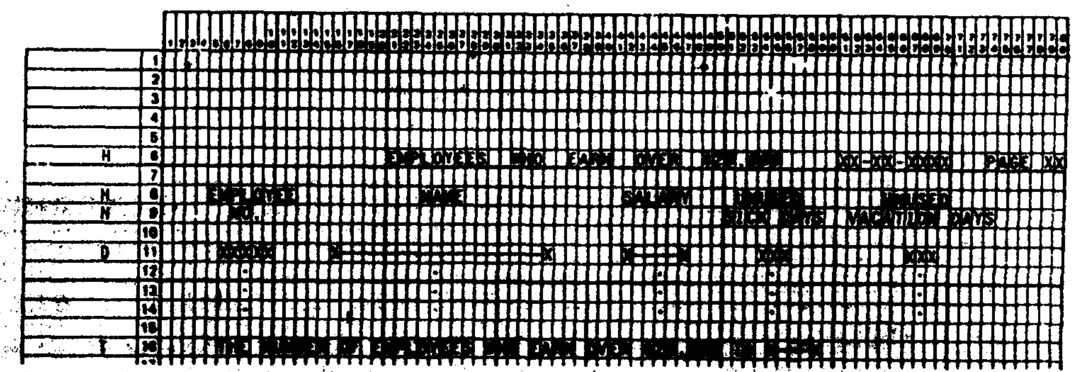


图 6.14 编程作业 2 的问题描述

3. 明细及一览表

磁盘文件的输入记录格式如下:

TerrNo, EmpName$, AnnualSalary

编写一个程序完成下列事项.

a. 打印每一个明细记录.

b. 在具有相同TerrNo的每组记录末，打印为该TerrNo支付的薪水总额. (假定记录按 TerrNo 顺序录入).

c. 运行结束时，打印所有薪水的总额.

4. 例外报表

假定已经为一种畅销计算杂志的每一订户建立了记录，记录格式如下:

CustName$, Address, CountryCode, SubscriptionCharse, SubscriptionPeriod

其中:

Country code 为: 1 = US; 2 = Canada。

Subscription period 为: 1 = 12个月; 2 = 24个月。

下列是有效的订阅费:

	12个月	24个月
US	**$ 25.66**	**$ 22.00**
Canada	**$ 40.75**	**$ 38.00**

编写一个程序，打印一份例外报表，该报表由收费有误的各订户的姓名、地址、国别及收费组成。

5. 一览表

一条收费路收费为：小轿车1.75美元，公共汽车2.00美元，8000磅以下的卡车3.50美元，8000磅以上的卡车5.00美元。

已为每辆已付通行费的车辆建立了文件记录，每条输入记录有一车辆类型字段，值为: c(小轿车)、b(公共汽车)、或者 t(卡车)。如果车辆类型为 t(卡车)，则重量字段为车辆的重量。

请打印一份一览表，读入文件，打印出各类车辆收费总计。

```
                              SUMMARY  REPORT    PAGR XX   XX-XX-XXXX

PASSENGER CAR TOTAL                               $  XX,XXX.XX
BUS TOTAL                                         $  XX,XXX.XX
TRUCKS  <  8000  LBS TOTAL                        $  XX,XXX.XX
TRUCKS  >=  8000  LBS TOTAL                       $  XX,XXX.XX

FINAL TOTAL COLLECTED                             $ XXX,XXX.XX
```

6. 明细表

编写一个程序，以描述利率对每月抵押付款额、利息总额及付款总额。请按列表方式打印输出。

输入文件记录格式如下:

MortageAmt, NoOfYears

打印每月付款额，利率范围为10%至17%(10%，11%，….17%)．准备自己的打印格式表，以便为每条输入记录单独产生一页．每一页应当标识为抵押利息表，抵押额及抵押年限作为单另的报表表头行，然后按10%到17%的利率打印每月抵押付款额、利息总额及付款总额，所有输出都应编辑．

第七章　控制中断过程：自顶向下模块化方法

本章要求：

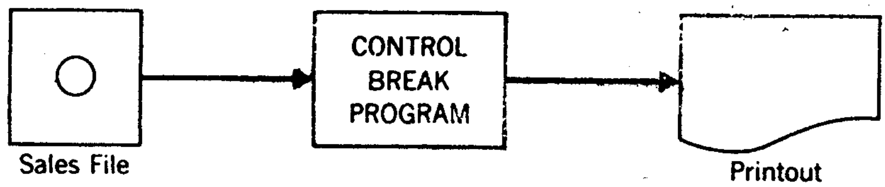
1. 熟悉细目和成组打印的附加特性．
2. 熟悉中断过程的简报．

§7.1　控制中断过程的介绍

§7.1.1　控制中断过程的例子

请看图 7.1 中的问题定义．一个销售报告的连续磁盘文件，每个报告有 3 个输入字段：一个是营业员所在部门号，一个是营业员号码，一个是营业员本周内销售额增长的数量，输入文件按部门号排序，因此所有与第 1 号部门有关的营业员记录均在第 2 号部门有关的营业员记录之前，如此等等．

系统流程图



销售记录格式

DeptNo, SalesNo, AmtOfSale

打印输出格式图

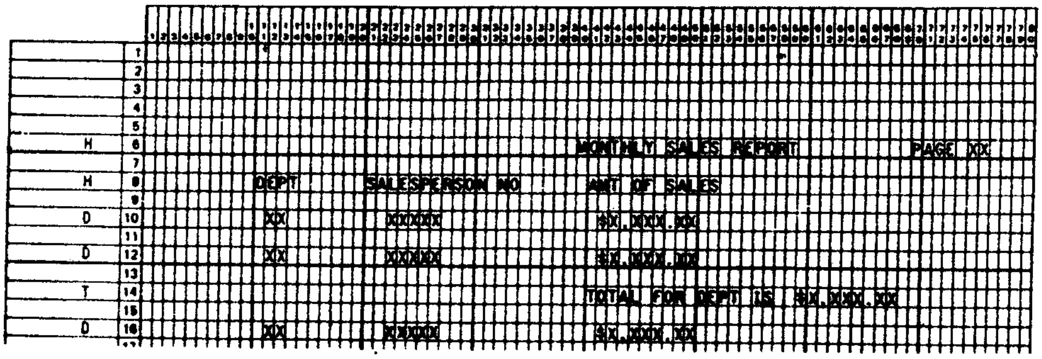


图 7.1　中断程序问题描述

也许有许多关于第 1 号、第 2 号等部门营业员的记录．也就是说，同一部门号可能有几个记录，这取决于给定部门中营业员的准确数量．输出报告不仅打印出每个营业员的销售量而且打印出每个部门的总销售量．

对于这个问题，细目打印是需要的；也就是说，每一个输入记录包含着一个营业员所有销售额；只有当系统分析员和程序员认为要看每个输入记录的数据时才推荐这样的细目打印．除了细目打印外，指出部门总数的总计也要打印，因此，在这个例子中，总计打印也是需要的，总计是针对每个部门的．

因此，当所有关于第 1 号部门的营业员记录已读和打印后，将打印一个关于第 1 部门的总计．类似地，当所有关于第 2 部门的营业员记录被读和打印后，将打印一个关于第 2 部门的总计，这种过程需要所有第 1 部门的记录先输入和处理，接着是下一部门的记录，如此下去．所以，这个输入连续文件必须依部门号排序．所有第 1 部门的营业员记录必须先输入，接着是第 2 部门的营业员记录，如此下去．另外，在每组的最后简单地累计和打印它将是不可能的（后面，我们将看到那些文件能被排入所需要的序列中）．

在每个输入记录被读入和打印后，细目打印以通常的方式进行．除了细目打印，每个输入记录的销售量增加到部门号总计中．当部门号发生变化时，将打印部门总计．既然部门号变化将导致部门总计的打印，那么把部门号叫做"控制字段"或"控制变量"．

因此，所有第 1 部门的营业员记录将输入和打印，而且要累计部门号总计．这个过程将持续下去直到读到一个部门号与前面那个不同的营业员记录．当读这个不同部门号的记录时，前面部门总计将打印出来．因此，第 1 个属于第 2 部门营业员的输入记录将导致一个第 1 部门总计的打印．既然当部门号变化后总计被打印，则称这种类型的过程为"中断过程"，其中部门号是控制字段．

这节给出关于中断过程的定义和由中断过程产生的输出实际说明．下一节集中讨论实际编码方法．用户可通过练习图 7.2 的伪代码工具为图 7.3 所示的程序做准备．

伪代码

```
START
    清屏
    PREFOR
        过程初始化
    ENDPERFORM
    将 DeptNo(部门号)存入 DeptHold
    PREFOR
        写标题
    ENDPERFORM
    DO WHILE 还有输入?
        IF DeptNo 发生变化 THEN
            PREFOR
                控制中断过程
            ENDPERFORM
        ENDIF
        PREFOR
            细目处理
        ENDPERFORM
        PREFOR
            读模块
        ENDPERRORM
    ENDDO
    PREFOR
        控制中断过程
    ENDPERFORM
    关闭输入文件
STOP
```

图 7.2 中断控制的伪代码

```
'This is a control break program
CLS
GOSUB InitializeModule
GOSUB ReadModule
LET HoldDept = DeptNo
GOSUB PrintHeadings
DO WHILE AreThereMoreRecords$ = "Yes"
        IF HoldDept <> DeptNo THEN
                GOSUB ControlBreak
        END IF
```

```
        GOSUB DetailProcessing
        GOSUB ReadModule
LOOP
GOSUB ControlBreak
CLOSE #1
END


ControlBreak:
        LPRINT
        LPRINT TAB(41);"TOTAL FOR DEPT IS ";
        LPRINT TAB(60);
        LPRINT USING "$#,###.##";DeptTotal
        LPRINT
        LPRINT
        DeptTotal = 0
        HoldDept = DeptNo
RETURN
DetailProcessing:
        IF LineCounter > 25 THEN
                GOSUB PrintHeadings
        END IF
        LPRINT TAB(12);
        LPRINT USING "##";DeptNo;
        LPRINT TAB(23);
        LPRINT USING "#####";SalesNo;
        LPRINT TAB(42);
        LPRINT USING "$#,###.##";AmtOfSale
        LPRINT
        LET LineCounter = LineCounter + 1
        LET DeptTotal = DeptTotal + AmtOfSale
RETURN


ReadModule:
        IF NOT EOF(1) THEN
                INPUT #1, DeptNo,SalesNo,AmtOfSale
        ELSE
                LET AreThereMoreRecords$ = "No"
        END IF
RETURN
PrintHeadings:
        LPRINT CHR$(12)
        LET PageNo = PageNo + 1
        LPRINT TAB(40);"MONTHLY SALES REPORT";
        LPRINT TAB(70);"PAGE";
        LPRINT TAB(75);
        LPRINT USING "##";PageNo
        LPRINT
        LPRINT TAB(11);"DEPT";
        LPRINT TAB(21);"SALESPERSON NO";
        LPRINT TAB(41);"AMT OF SALES"
        LPRINT
        LET LineCounter = 0
RETURN


InitializeModule:
        OPEN "Sales.Dat" FOR INPUT AS #1
        LET AreThereMoreRecords$ = "Yes"
        LET PageNo = 0
RETURN
```

图 7.3 Turbo Basic 中断控制程序

上述中断过程的输入见图 7.4，输出也见此图。

样本输入

```
1,12345,988.5499877929688
1,12346,3537
1,12347,34.9900016784668
2,12222,9877
2,12234,87.76999664306641
3,15645,980
3,12321,1987
4,12999,1343.300048828125
4,16732,1779
4,16437,4939.08984375
```

样本输出

```
                                  MONTHLY SALES REPORT            PAGE  1

DEPT        SALESPERSON NO         AMT OF SALES

  1            12345               $   988.55

  1            12346               $3,537.00

  1            12347               $    34.99

                                   TOTAL FOR DEPT IS  $4,560.54

  2            12222               $9,877.00

  2            12234               $    87.77

                                   TOTAL FOR DEPT IS  $9,964.77

  3            15645               $   980.00

  3            12321               $1,987.00

                                   TOTAL FOR DEPT IS  $2,967.00

  4            12999               $1,343.30

  4            16732               $1,779.00

  4            16437               $4,939.09

                                   TOTAL FOR DEPT IS  $8,061.39
```

图 7.4 中断程序输入和输出样例

层次图复习

图 7.5 中的层次图指出子程序中所用于过程的名字和它们执行的模块. 图中缺角的模块或子过程表示这些模块或子程序不只在程序的一处执行。

像图 7.5 这样的层次图, 对于有多个子过程的程序中表明模块间关系是很有用的. 第十一章讨论其它类型模块时, 将更细致地讨论这个问题.

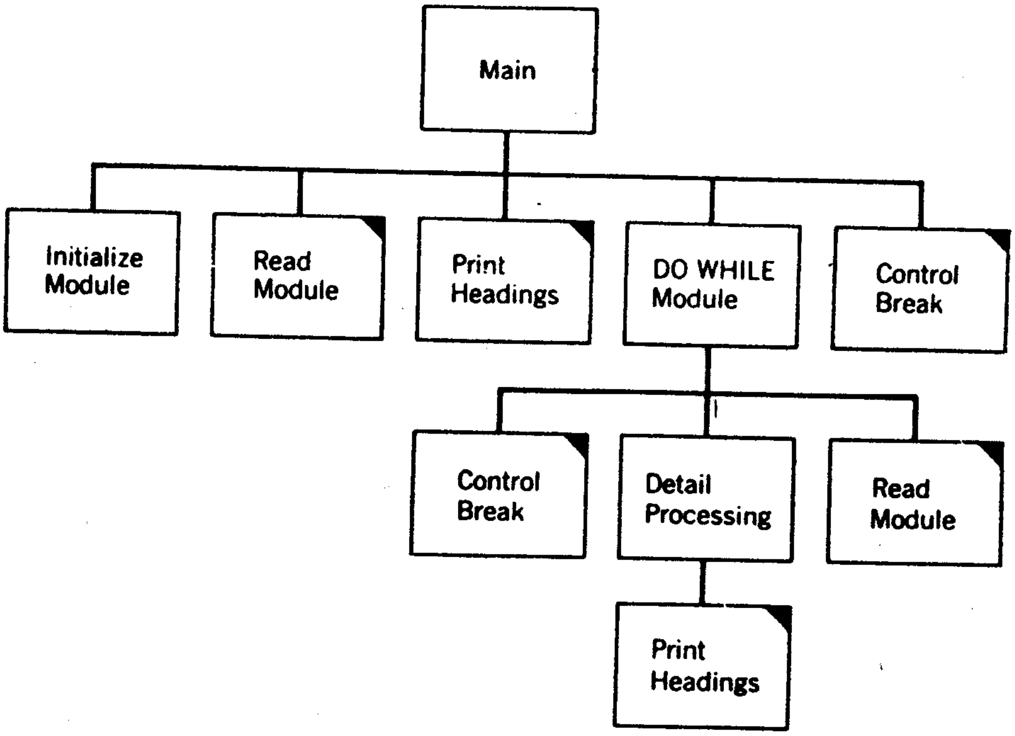


图 7.5　中断控制程序层次图

§7.1.2　单级控制中断

§7.1.2.1　概述

先清屏并执行初始化模块, 这个初始化模块打开输入文件并初始化 AreThereMoreReconds$ 和 PageNO 这两个变量; 然后完成一个从 ReadModule (读模块) 中的初始读入.

将第 1 个记录的部门号存入一个称作 HoldDept 的保持区域, 并打印标题. 现在, 讨论 DO WHILE 模块.

读每一个输入记录时, 将首先测试, 看部门号是否已发生变化, 执行控制中断; 然

后，再执行细目过程；如果部门号没有变化，则只执行细目过程．否则，读下一记录；细目过程模块也在 DO WHILE … LOOP 中执行．这个模块打印一个细目行，维护一个行计数器，将 AmtOfSale 加到 DeptTotal 上．

如前所述，在 DO WHILE … LOOP 中，一开始就有一个关于部门号是否变化的测试；而只有当第 1 个带有新部门号的记录被读入时，中断控制模块才执行．

在控制中断模块中，打印总计行：先重新初始化部门总计，将其置为零；并将新的部门号存入 HoldDept 之中；然后返回 DO WHILE，完成了第 1 个带有新部门号记录的细目处理．

总之，当部门号发生变化时，DO WHILE 处理一个控制中断子过程；没发生变化时，在细目处理中处理每个记录，然后读下一记录．

这个程序叫做单级控制中断程序．因为只有一个字段一部门号，所以它引起总计打印．为实现单级控制中断过程，必须在 DO WHILE 之前完成下列编码步骤：

1. 初始化过程——打开输入文件初始化变量．
2. 第一个输入记录，被读入输入模块．
3. 第一个记录的控制字段（部门号）必须被移入保持区域以用于比较，这个存有部门号的保持区域叫 HoldDept．
4. 打印标题．

具体的伪代码和程序

DO WHILE … LOOP 前的指令
⋮

下面是伪代码和 Turbo Basic 程序：

伪代码节选

```
清屏
PERFORM
    初始化程序
ENDPERFORM
PERFORM
    读模块
ENDPERFORM
将部门号存入 DpetHold
PERFORM
```

```
    写标题
ENDPERFORM
DO WHILE 只要还有一个输入
      ⋮
      ⋮
ENDDO
```

Turbo Basic 程序段

```
CLS
GOSUB InitializeModule
GOSUB ReadModule
LET HoldDept = DeptNo
GOSUB PrintHeadings
DO  WHILE AreThereMoreRecords$ = "Yes"
      ⋮
LOOP
```

在伪代码中的 PERFORM … ENDPERFORM 指出需要模块或单独程序，它也可用 Turbo Basic 中的 GOSUB 编写.

在控制中断过程中自顶向下编程

可以用 GOSUB 语句帮助编制自顶向下模块程序. 注意上面的节选不包括具体需要的初始化，读和打印标题. 我们将这些具体的东西留给子程序，重点放在编码时的程序结构，而不是特殊指令的语法；此外，GOSUB 读模块和 GOSUB 打印标题减少重复编码. 用户将在后面看到：读和写标题在程序中也是必需的，所以最好将它们留给子程序. 这些子程序可在程序中的不同地方使用.

§7.1.2.2 DO WHILE … LOOP 循环

通过比较 DeptNo 和 Hold Dept 来开始 DO WHILE … LOOP 循环. 第一段时间 HoldDpet 和 DeptNo 相等，因为我们只是在进入循环前把 DeptNo 移入 HoldDpet. 当所有子序列通过循环时，HoldDept 将与刚读入的部门号比较；当两者相等时，就没必要执行控制中断程序. 因此，除非一个控制中断或部门号变化发生，将不作控制中断. 当每一个输入记录读完，将执行细目处理.

伪代码节选

```
DO  WHILE  只要还有输入
  :        (控制中断测试)
  ┌──────────────────────────┐
  │ PERFORM                  │
  │     DetailProcessing     │
  │ ENDPERFORM               │
  └──────────────────────────┘
  : }  — 读入下一记录
ENDDO
```

§ 7.1.2.3 DetailProcessing(细目处理)模块

```
DetailProcessing:
    IF  LineCounter > 25 THEN
        GOSUB PrintHeadings
    END IF
    LPRINT TAB(12);
    LPRINT USING "**"; DeptNo;
    LPRINT TAB(23);
    LPRINT USING "*****"; SalesNo;
    LPRINT TAB(42);
    LPRINT USING "$*,***.**"; AmtOfSale
    LPRINT
    LET LineCounter = LineCounter + 1
    LET DeptTotal = DeptTotal + AmtOfSale
RETURN
```

DetailProcessing(细目处理)模块

a. 看页尾是否到了；如果到了，就执行打印标题模块，即在新页上打印标题.
b. 打印一个包含DeptNo，SalesNo，和Amt ofSale的细目行.
c. 增加Line Counter.
d. 将AmtofSale加到DpetTotal.

在细目处理后，DO WHILE … LOOP 执行读入下一个记录的读模块.
继续以此方式处理输入记录，直到存于 HoldDept 中的输入记录与前一记录部门号不

同为止。当它们不同时，发生一个控制中断。因此，当 DeptNo 与 HoldDept 不等时，执行一个打印累计部门总计的子程序。

控制中断子程序打印前一部门的总计。当打印了总计并重新初始化为零，且新的部门号装入 HoldDept 中时，转入紧跟 END IF 后的 DO WHILE 循环的下一步；通过继续执行细目处理模块中的当前记录来继续下去。这包括打印一个细目行，增加 LineCounter，并将 AmtofSale 加到 DeptTotal 之中；正如前面所作的那样（在此模块中，也测试了页尾）。

完整的 DO WHILE … LOOP 如下：

伪代码节选

```
DO  WHILE  只要还有输入
    IF  部门号发生变化
        PERFORM
            :
            :   }  在此处编写控制中断伪代码
            :
        ENDPERFORM
    ENDIF
    PERFORM

       细目过程
    ENDPERFORM
    PERFORM
        读模块
    ENDPERFORM
ENDDO
```

程序节选

```
DO  WHILE AreThereMoreRecords$ = "Yes"
    IF  HoldDept <> DeptNo THEN
        GOSUB ControlBreak
    END IF
    GOSUB DetailProcessing
    GOSUB ReadModule
LOOP
```

如果部门号有变化，则在执行的 Turbo Basic 程序中此模块叫控制中断。如无变化，当前记录被打印并且其值在新记录读之前被加到 Dept Total 上。

§7.1.2.4 控制中断模块

当一个输入记录的控制字段——部门号与存在 HoldDept 中的不同时中断控制模块执行。正如我们所见到的一样，HoldDept 包含前一记录的部门号；当部门号改变时，必须:

a. 打印一行在DeptTotal中的部门总计。
b. 重新初始化DeptTotal，即控制总计，保证下一部门总计从零开始累计。
c. 将部门号当前值移入HeldDept中，使之能将后面的输入记录与新的部门号控制字段比较。
d. 继续DO WHILE … LOOP循环，打印细目行，增加控制总计，读下一记录。

下面的控制中断过程包含所需处理:

伪代码节选

```
DO  WHILE  只要还有输入
    IF  部门号发生变化  THEN
        PERFORM
          :
          :   }  控制中断程序
          :
        ENDPERFORM
    ENDIF
      :
ENDDO
```

程序节选

```
DO  WHILE AreThereMoreRecords$ = "Yes"
    IF  HoldDept <> DeptNo THEN
        GOSUB ControlBreak
    END IF
    .
    .
    .
LOOP
  .
  .
  .
 ControlBreak:
     LPRINT
     LPRINT TAB(41); "TOTAL FOR DEPT IS";
     LPRINT TAB(60);
     LPRINT USING "$*,***.**"; DeptTotal
     LPRINT
     LPRINT
     DeptTotal = 0
     HoldDept = DeptNo
 RETURN
```

在这里，可以保存一个叫 PrintTotals 的独立模块，用以打印总计的细目；其编码如下:

```
ControlBreak:
    GOSUB PrintTotals
    DeptTotal = 0
    HoldDept = DeptNo
RETURN
PrintTotals:
    LPRINT
    LPRINT TAB(41); "TOTAL FOR DEPT IS ";
    LPRINT TAB(60);
    LPRINT USING "$*,***.**"; DeptTotal
    LPRINT
    LPRINT
RETURN
```

因为 ControlBreak 子程序是从 DO WHILE … LOOP 调用出来的，所以在 ControlBreak 程序最后的 RETURN 将把控制返回到 DO WHILE … LOOP 中的下一条指令。在循环中，下一条指令是:

```
GOSUB  Detailprocessing
```

在此，用新输入的数据打印细目行，当前值被加到新的总计中，总计最初被置为 0。

§ 7.1.2.5 DO WHILE … LOOP 后过程

到目前为止，我们已经看到 DO WHILE … LOOP 是如何执行的；当一个与前一记录部门号相同的记录读入时，打印该记录并将 AmtOfSale 加到 DeptTotal 上。

当部门号发生变化时，DeptNo 和 HoldDept 值不同了，则执行 Control Break 子程序。在 ControlBreak 程序中，打印总计行、重新初始化 DeptTotal 后置零、将当前的 DeptNo 存入 HoldDept；然后，返回 DO WHILE … LOOP 循环中执行 DetailProcessing 的地方；打印当前输入记录（这是一个新控制组中的第一个记录），将其值加到一个新的 DeptTotal。当文件结束条件成立时，即是过程所需要的。

首先，必须在最后的记录读入并处理后，打印最后的控制总计。当一个有新控制字段的记录读入时，打印控制中断总计。然后，最后一组记录的总计当条件 AreThereMoreReconds $ = "No"满足时累计，但不打印控制总计，因为没有引发变化的子序列记录。考虑如下情况：

```
DeptNo
  01
  01
  01
--------
  02      ←  当 02 读入时，打印 01 总计
  02
  02
  02
--------
  03      ←  当 03 读入时，打印 02 总计
  03
--------
  04      ←  当 04 读入时，打印 03 总计
  04
  04
--------
 /AT  END        在此处，04总计必须在AreThereMoreRecords $ 被置
                 为"NO"时中断。
```

在这之前，必须有一个打印最后第 04 号记录及 04 总计的过程。因此，在主模块中，在 DO WHILE … LOOP 执行完及 AreThereMoreRecords $ = "Yes"时，必须对最终总计的打印进行人工干预。

伪代码节选

```
START
    :
    :
    DO  WHILE  没有输入
        :
        :
    ENDDO
   ┌──────────────────┐
   │ PERFORM          │
   │     控制中断程序  │
   │ ENDPERFORM       │
   └──────────────────┘
   关闭输入文件
STOP
```

Turbo Basic 节选

```
   :
   DO  WHILE AreThereMoreRecords$ = "Yes"
       :
   LOOP
 ┌─────────────────────┐
 │GOSUB ControlBreak   │
 └─────────────────────┘
   CLOSE #1
   END
```

请注意: 在DO WHILE ··· LOOP之后，又写GOSUB ControlBreak，以便能处理最后一个记录及打印最后控制总计。

如上所述，带有细目打印的单级控制中断过程的全程序参见图 7.3. 程序主体中包含专用模块，每一模块完成一个特殊功能. 按这种组织方式，程序的全部结构出现在顶部，细目留给子模块，这叫做"自顶向下程序"。这些专用模块被编写子程序，放在 END 语句后面.

§7.1.2.6 读模块

在最后一个记录读完后马上有 EOF 条件成立。因此下面这样编码就是错误的，因为

最后一个记录没有处理:

错误的编码

```
INPUT ...
.
.
.
DO  WHILE NOT EOF(1)
     .
     .
     .
    INPUT ...
LOOP
```

确信当最后一个记录被读入并处理后，结束 DO WHILE … LOOP，编码如下:

```
DO  WHILE AreThereMoreRecords$ = "Yes"
    .
    .
    .
    GOSUB ReadModule
LOOP
    .
    .
    .
ReadModule:
    IF  NOT EOF(1) THEN
        INPUT ...
    ELSE
        LET AreThereMoreRecords$ = "No"
    END IF
```

当最后一个记录被读入并处理后，串变量 AreThereMoreRecords $ 被置为“NO”

§ 7.1.3 精确表达以提高控制中断报告的质量

此处讨论的精确表达用图 7.6 来说明。它是一个将上面讨论的方法和精确表达结合在一起的单级控制中断程序。

```
'This is an example of a control break program
CLS
GOSUB InitializeModule
GOSUB ReadModule
LET HoldDept = DeptNo
GOSUB PrintHeadings
'*************************************************************
' The DO WHILE determines if a control break has occurred
' and processes records accordingly
'*************************************************************
DO WHILE AreThereMoreRecords$ = "Yes"
        IF HoldDept <> DeptNo THEN
                GOSUB ControlBreak
        END IF
        GOSUB DetailProcessing
        GOSUB ReadModule
LOOP
```

```
GOSUB ControlBreak
LPRINT
LPRINT TAB(41); "THE FINAL TOTAL IS ";
LPRINT TAB(60);
LPRINT USING "###,###.##"; FinalTotal
CLOSE #1
END
'***********************************************************
' The control break module prints total when a change in
' DeptNo occurs
'***********************************************************

ControlBreak:
        IF HoldDept > DeptNo THEN  'This is a sequence error
                PRINT "Sequence Error"
        END IF
        LPRINT
        LPRINT TAB(41); "TOTAL FOR DEPT IS ";
        LPRINT TAB(60);
        LPRINT USING "$#,###.##"; DeptTotal
        LPRINT
        LPRINT
        LET FinalTotal = FinalTotal + DeptTotal
        DeptTotal = 0
        HoldDept = DeptNo
        GOSUB PrintHeadings
RETURN
'***********************************************************
' The DetailProcessing module prints output for each line
' of input
'***********************************************************
DetailProcessing:
        IF LineCounter > 25 AND AreThereMoreRecords$ = "Yes" THEN
                GOSUB PrintHeadings
        END IF
        LPRINT TAB(12);
        LPRINT USING "##"; DeptNo;
        LPRINT TAB(23);
        LPRINT USING "#####"; SalesNo;
        LPRINT TAB(42);
        LPRINT USING "$#,###.##"; AmtOfSale
        LPRINT
        LET LineCounter = LineCounter + 1
        LET DeptTotal = DeptTotal + AmtOfSale
RETURN
'***********************************************************
' TheReadModule reads input records and determines if an
' end of job has been reached
'***********************************************************
ReadModule:
        IF NOT EOF(1) THEN
                INPUT #1, DeptNo, SalesNo, AmtOfSale
        ELSE
                LET AreThereMoreRecords$ = "No"
        END IF
RETURN
'***********************************************************
' The PrintHeading module prints new headings at the
' beginning of the run, when more than 25 output lines have
' printed and when a control break occurs
'***********************************************************
PrintHeadings:
        LPRINT CHR$(12)
        LET PageNo = PageNo + 1
        LPRINT TAB(40); "MONTHLY SALES REPORT";
        LPRINT TAB(70); "PAGE";
        LPRINT TAB(75);
        LPRINT USING "##"; PageNo
        LPRINT
        LPRINT TAB(11); "DEPT";
        LPRINT TAB(21); "SALESPERSON NO";
        LPRINT TAB(41); "AMT OF SALES"
        LPRINT
        LET LineCounter = 0
RETURN
```

```
'***********************************************************
' The InitializeModule opens the input file and initializes
' specific storage areas
'***********************************************************
InitializeModule:
        OPEN "Sales.Dat" FOR INPUT AS #1
        LET AreThereMoreRecords$ = "Yes"
        LET PageNo = 0
        LET FinalTotal = 0
RETURN
```

图 7.6(a) 精确表达的单级控制中断程序

```
                                  MONTHLY SALES REPORT          PAGE  1
DEPT        SALESPERSON NO        AMT OF SALES
  1            12345                 $988.55
  1            12346               $3,537.00
  1            12347                  $34.99

                                  TOTAL FOR DEPT IS   $4,560.54
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                  MONTHLY SALES REPORT          PAGE  2
DEPT        SALESPERSON NO        AMT OF SALES
  2            12222               $9,877.00
  2            12234                  $87.77

                                  TOTAL FOR DEPT IS   $9,964.77
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                  MONTHLY SALES REPORT          PAGE  3
DEPT        SALESPERSON NO        AMT OF SALES
  3            15645                 $980.00
  3            12321               $1,987.00

                                  TOTAL FOR DEPT IS   $2,967.00
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                  MONTHLY SALES REPORT          PAGE  4
DEPT        SALESPERSON NO        AMT OF SALES
  4            12999               $1,343.30
  4            16732               $1,779.00
  4            16437               $4,939.09

                                  TOTAL FOR DEPT IS   $8,061.39


                                  THE FINAL TOTAL IS  $25,553.70
```

图 7.6(b) 打印输出

下面讨论上述程序所作的精确表达:

§7.1.3.1 打印最后总计

有时，控制中断处理也需要打印包含最后总计的总计行。它是在最后控制总计写出后才打印的。

方法 1

称作 FinalTotal 的最后总计必须在每个记录增加 AmtOfSale 时累计。这可通过在图 7.3 中的 DetailProcessing 模块中增加如下赋值语句的方法来实现:

```
LET  FinalTotal  =  FinalTotal +AmtOfSale
```

方法 2

累计 FinalTotal 可被在 ControlBreak 模块或子程序中加入每个 DeptTotal，即 FinalTotal 并不对每一个细目记录都积累，而只是当控制中断发生时才积累，这将通过在重新初始化 DeptTotal 前编码来完成:

```
ControlBreak:
     .
     .
     .
     LET FinalTotal = FinalTotal + DeptTotal
     LET DeptTotal = 0
     .
     .
     .
RETURN
```

注意，第 2 种方法比第 1 种方法有效得多. 假设有 10,000 个输入记录，但只有 20 个部门控制中断. 如果在 DetailProcessing 中对每一个输入记录的 AmtofSale 加到 FindTotal 上，将有 10, 000 个增量，或者当每个中断控制发生时增加 DeptTotal 到 FindTotal 上，则得到 20 倍的增量。因此，增加 DeptTotal 到 FindTotal 上将导致比增加 AmtofSale 到 FindTotal 上少得多的增量。

§7.1.3.2 在每个控制中断之后开始新的一页

可能控制中断报告会分散到许多用户。例如，部门 01 的列表如上说明，可能会传到部门 01 的用户那; 部门 02 的列表可在会传到那个部门，如此等等。在这种情况下，有必要将每个部门的数据从一个新页开始，这样，控制域中的一个变化将也包含一条语句，执行 PrintHeadings 子程序，使控制中断发生时纸进到新的一页。我们将在 ControlBreak 中增加 GOSUB 语句。使每次这个模块执行时在新页头打印标题。

图 7.6 打印每个输入记录的部门号在每个细目行上，这有些冗余，而在每页开头打印一次会更好些。

```
                MONTHLY STATUS REPORT     PAGE xx

DEPT-xx

              SALESPERSON NO                AMT OF SALES

                  12345                     $7,326.45
                  18724                     $9,264.55

                               TOTAL FOR DEPT IS $16,591.00
```

§7.1.3.3 序列检验或排序：确认输入数据的顺序

对于准确的控制中断处理，记录必须按控制字段排序，看下面这个序列的错误：

```
DetpNo
    01
    01
    02    ←—序列错——DeptNo 02 输入顺序不对
    01
    01
```

因为输入序列错误是有可能发生的，因此有必要在每个中断控制之后检查加以确认当前 DeptNo 比在 Holc Dept 中的前一值大。如果当前 DeptNo 小于 HoldDpet，则有序列错发生，且打印出错误信息。在这种情况下，希望结束处理。

另一种确认序列正确的方法是：用计算机排序。用计算机排序的文件序列是正确的，所以具有这种文件的处理不需要单独的序列检查程序。用户可以编自己的排序过程，但这样太复杂了；或者，用一个已编好的排序工具，象在 Turbo Basic 例子盘中的程序一样。还有许多其它排序工具，最简单的是由 DOS 提供的；其格式如下：

```
A> SORT <未排序文件名> 排序后的文件名
```

例：

```
A> SORT <Unsorted.Dat> Sorted.Dat
```

DOS 保证文件中每一记录的第 1 个字段是按所希望的顺序排定的．如确信在顺序文件中每个记录都是有序的，最好在执行控制中断程序之前对文件进行排序．

一旦文件被排序了，可认为在排序文件中的所有记录均有正确的序．因此，序列检查过程只对手工排序输入文件是必要的（手工排序输入文件是由负责在输入数据时对文件进行排序的操作员在磁盘上排定的）．

图 7.6 给出了在 ControlBreak(控制中断)模块中的序列检查．

§ 7.1.3.4 有更多的数据时才打印标题

在遇到页结束条件之后打印标题，因此有必要只在 AreThereMoreReconds $ 仍等于"Yes"时打印它们．否则，最后一页报告将产生另一标题，这是不对的．

页结束条件的测试将导致：只有当 LineCounter > 25 和 AreThereMoreRecords $ = "Yes"时，才执行标题程序：

```
DetailProcessing:
    IF  LineCounter > 25 AND AreThereMoreRecords$ = "Yes" THEN
        GOSUB PrintHeadings
    END IF
    .
    .
    .
RETURN
```

完整的单级控制中断程序包含所有图 7.6 中所述的精确描述

§ 7.1.4 单级中断过程小结

单级控制中断问题步骤小结

1. 打开输入文件，初始化变量
2. 读初始记录
3. 将第1个记录的控制字段移入保护域．
4. 打印标题．
5. 当控制字段等于保护域时处理输入记录，也就是：在控制总计中增加一合适值．在控制中断过程中细目打印有时是必要的，但它经常省略．如果需要，细目处理程序将打印细目记录，增加行记数器值，测试是否为页末．
6. 如果控制字段与保护域不相等：

   打印控制总计
   置控制总计初值为零

用一个新的控制字段值重新初始化保持字段.

象第 5 步那样处理细目记录

当每个控制总计出现在不同页上时打印新标题.

7. 当所有记录被读入并处理后，完成一个打印最后控制总计的控制中断.
8. 打印需要的最后总计.
9. 关闭输入文件.

自我测验

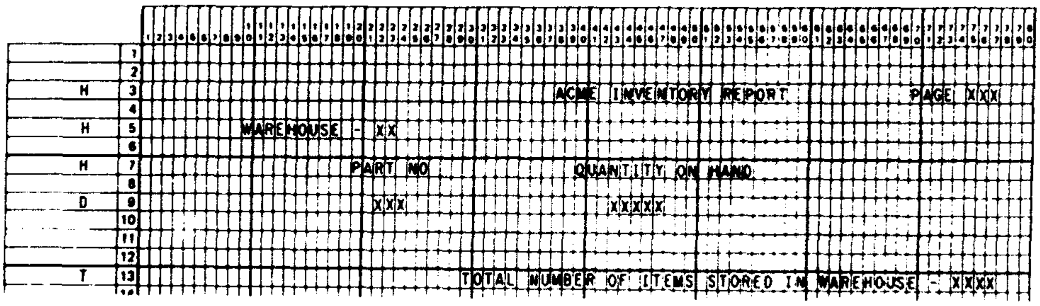
1. 当特殊输入字段变化引起总计打印，这个过程叫做________.
2. 在第1个问题中，决定打印的那个字段叫________.
3. 在第1个问题中，记录必须按________来排序.
4. 在控制中断过程中，通常移动或指派控制字段到________，在读完第1个记录之后.
5. 在控制中断过程中，通常把________作为DO WHILE … LOOP中的第1条指令.
6. 如果一个输入控制字段与存在保持域中的控制字段相等，那么________.
7. 如果一个输入控制字段与存在保持域中的控制字段不等，那么________.
8. 当遇到EOF条件之后，必须________.
9. 如果每一控制组以单独页开始，必须在________模块执行标题程序.
10. 如果需要最后总计，最有效的办法是在________模块中累计最后总计.
11. 在控制中断模块，必须打印________，初始化________置为零，并移动________.
12. 当每个输入记录导致输出行打印时，称之为________.

考虑下面的问题描述（问题 13～15）：

由磁盘记录组成的输入字段

Warehouse, PartNo, QtyOnHand

ReportOut打印格式图



---

13. 为上面的问题描述编码。
14. 为上面的问题描述编写打印细目行的程序。
15. 为上面的问题描述编写ControlModule模块。

§7.2 多级控制中断

应该记住，文件必须按控制字段排序以完成控制中断处理。假设需要两个字段做为控制字段。考虑下面处理或细目输入文件：

交易数据

DeptNo, SalesNo, AmtOfSale

营业员每做成一笔交易，便建立一个表示其部门号(DeptNo)，营业员号(SalesNo)和营业额(AmtOfSale)的记录。在这种情况下，不同于以前所述，如营业员在给定时间做成许多交易，则有关于该营业员的多于一个的记录；即，如果第1部门的第1号营业员售3种货物，将有3个关于这个营业员的输入记录；如果第2部门的2号营业员销售4种货物，将有4个关于这个营业员的记录，如此等等。

这种输入与前述例子中的输入有如下不同之处：每个营业员属于一个特殊部门且可能有许多记录；每销售一种货物对应一个记录；输入文件排序方法是所有部门1的营业员的记录排在第1位，然后是部门2的营业员记录，如此下去。其中，每一部门中第1号营业员的记录排在前面，随后为第2号营业员的记录，如此等等。下面是期望的输入：

DeptNo	SalesNo		AmtOfSale
01	004	In sequence within DeptNo 01	127.23
01	004		100.14
01	006		027.45
01	006		052.23
01	006		126.67
01	008		223.28
02	003	In sequence within DeptNo 02	111.14
02	003		027.23
02	003		119.25
02	005		600.45
02	018		427.33
03	014		100.26
.	.		.
.	.		.
.	.		.

注：上图中SalesNo栏的说明分别为：部门1有序排列，和部门2有序排列。

那么，有如下两个控制字段：

营业员号——辅助控制字段
部　门　号——主控制字段

在同一部门号内记录按营业员号排序，对于给定的部门号中的某一营业员有许多记录。

假定希望打印如图 7.7 打印格式表中所指出的部门报告。

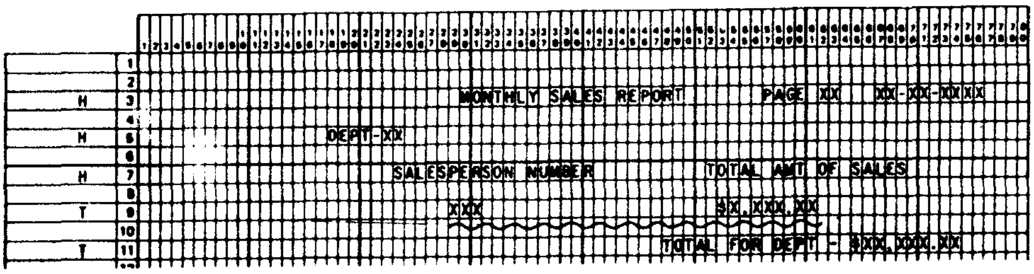


图 7.7　多级中断控制报告的打印格式图

在打印每一行前，对每个营业员累计其销售量。因此，如果部门 1 的 1 号营业员销售 3 种货物，输入到不同的输入记录中，在三个记录均被读入后打印一行。

这里不需要细目打印，而程序产生营业员总计和部门总计的打印。营业员总行打印是在所有给定营业员的记录全部处理后才进行的。而且，当部门号中断发生时才打印下一部门号的部门总计和标题。

在主模块中，先执行初始化程序，即打开文件、存取数据、执行读模块、存控制字段，然后执行标题程序．既然第 2 个标题行包括一个文字上的“部门号”和实际的“部门号”，则必须在初始读入之后打印标题．在图 7.3 中的第 1 个控制中断过程中，在输入数据之前打印标题，这是因为不需要第一个部门的记录作为标题打印的一部分．

首先，检验用于准备这个程序的计划工具：伪代码在图 7.8 中、层次图见图 7.9、Turbo Basic 程序见图 7.10．

```
START
    PERFORM
        Open the input file
        Store the Date
        PERFORM
            Read a Record
        ENDPERFORM
        Store Control Fields
    ENDPERFORM
    PERFORM
        Write the Headings
    ENDPERFORM
    DO  WHILE there is still data
        IF  there is a DeptNo change THEN
            PERFORM
                PERFORM
                    Move SalesNoTotal to Output Area
                    Write a Record
                    Add SalesNoTotal to DeptNoTotal
                    Initialize SalesNoTotal
                    Store SalesNo
                ENDPERFORM
                PERFORM
                   Write DeptNoTotal
                ENDPERFORM
                Initialize DeptNoTotal
                Store DeptNo
                PERFORM
                    Write Headings
                ENDPERFORM
            ENDPERFORM
        ENDIF
        IF  there is a SalesNo change
        THEN
            PERFORM
                Move SalesNoTotal to Output Area
                Write a Record
                Add SalesNoTotal to DeptNoTotal
                Initialize SalesNoTotal
                Store SalesNo
            ENDPERFORM
        ENDIF
        Add Amt to a Total
        PERFORM
            Read a Record
        ENDPERFORM
    ENDDO
    PERFORM
        Dept break processing
    ENDPERFORM
    PERFORM
        END-of-Job Operations
    ENDPERFORM
STOP
```

图 7.8　多级控制中断处理伪代码

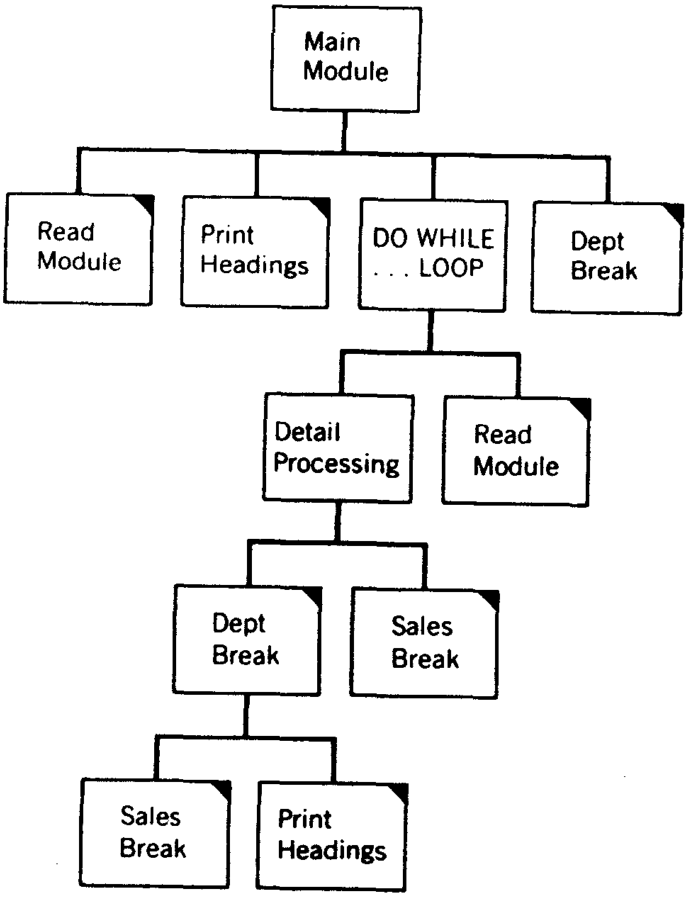


图 7.9　多级控制中断处理层次图

〔注〕: SalesBreak, DeptBreak, PrintHeadings和ReadModule框缺角是因为它们都是从两个不同模块中执行的.

```
 This is an example of a Multiple Control Break Program
'***********************************************************
' The following instructions are part of an
' initialization routine
'***********************************************************
OPEN "Sales2.Dat" FOR INPUT AS #1
DateOfRun$ = DATE$
AreThereMoreRecords$ = "Yes"
GOSUB ReadModule
LET HoldDept = DeptNo
LET HoldSalesNo = SalesNo
GOSUB PrintHeadings
'*************************************************
' The DO WHILE controls the overall
' processing of the program
'*************************************************
DO WHILE AreThereMoreRecords$ = "Yes"
        GOSUB DetailProcessing
        GOSUB ReadModule
LOOP
GOSUB DeptBreak
CLOSE #1
END
```

```
'*******************************************
' The Read Module reads from the input file
' and determines if an end of file condition
' has occurred
'*******************************************
ReadModule:
        IF NOT EOF(1) THEN
                INPUT #1, DeptNo, SalesNo, AmtOfSale
        ELSE
                LET AreThereMoreRecords$ = "No"
        END IF
RETURN
'*********************************************
' The PrintHeadings module prints headings on a
' new page at the beginning and whenever a page
' overflow has occurred
'*********************************************
PrintHeadings:
        LPRINT CHR$(12)
        LET PageNo = PageNo + 1
        LPRINT TAB(30); "MONTHLY SALES REPORT";
        LPRINT TAB(57); "PAGE";
        LPRINT TAB(62);
        LPRINT USING "##"; PageNo;
        LPRINT TAB(67); DateOfRun$
        LPRINT
        LPRINT TAB(18); "DEPT-";
        LPRINT TAB(23);
        LPRINT USING "##"; DeptNo
        LPRINT
        LPRINT TAB(24); "SALESPERSON NUMBER";
        LPRINT TAB(53); "TOTAL AMT OF SALES"
        LPRINT
RETURN
'*********************************************
' The Detail Processing module processes detail
' records
'*********************************************
DetailProcessing:
        IF DeptNo <> HoldDept THEN
                GOSUB DeptBreak
        END IF
        IF SalesNo <> HoldSalesNo THEN
                GOSUB SalesBreak
        END IF
        LET SalesTotal = SalesTotal + AmtOfSale
RETURN
'*********************************************
' The Dept Break module is executed when there
' is a change in DeptNo
'*********************************************
DeptBreak:
        GOSUB SalesBreak
        GOSUB PrintDeptTotalLine
        DeptTotal = 0
        IF AreThereMoreRecords$ = "Yes" THEN
                HoldDept = DeptNo
                GOSUB PrintHeadings
        END IF
RETURN
'*********************************************
' The Sales Break module is executed when there
' is a change in SalesNo
'*********************************************
SalesBreak:
        GOSUB PrintSalesLine
        DeptTotal = DeptTotal + SalesTotal
        SalesTotal = 0
        HoldSalesNo = SalesNo
RETURN
```

```
'*********************************************
'PrintSalesLine prints a line when a SalesNo
' break occurs
'*********************************************
PrintSalesLine:
        LPRINT TAB(30); HoldSalesNo
        LPRINT TAB(53);
        LPRINT USING "$#,###.##"; SalesTotal
RETURN
'*****************************************
' PrintDeptTotalLine prints a line when there
' is a change in DeptNo
'*****************************************
PrintDeptTotalLine:
        LPRINT
        LPRINT TAB(48); "TOTAL FOR DEPT - ";
        LPRINT TAB(65);
        LPRINT USING "$$#,###.##"; DeptTotal
        LPRINT
RETURN
```

图 7.10(a) 多级控制中断处理的 Turbo Basic 程序

```
              MONTHLY SALES REPORT        PAGE  1   06-13-1990

DEPT- 1

     SALESPERSON NUMBER            TOTAL AMT OF SALES

            4
                                   $  227.37
            6
                                   $  205.95
            8
                                   $  223.28

                              TOTAL FOR DEPT -      $656.60
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

              MONTHLY SALES REPORT        PAGE  2   06-13-1990

DEPT- 2

     SALESPERSON NUMBER            TOTAL AMT OF SALES

            3
                                   $  257.63
            5
                                   $  600.45
            18
                                   $  427.33

                              TOTAL FOR DEPT -   $1,285.41
~~~~~~~~~~~~~~~~~~~~~~~~~~~~  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

B22

```
                MONTHLY SALES REPORT        PAGE  3   06-13-1990
DEPT- 3
      SALESPERSON NUMBER          TOTAL AMT OF SALES
             14
                                   $  100.26
                             TOTAL FOR DEPT -    $100.26
```

图 7.10(b)　打印输出

在这个程序中用两级中断控制过程，需要两个保持域，为不同目的，一个为部门号，一个为营业员号。

伪代码节选

```
START
    PERFORM
        打开输入文件
        存数据
        PERFORM
            读记录
        ENDPERFORM
        存控制字段
    ENDPERFORM
    PERFORM
        写标题
    ENDPERFORM
    DO WHILE  只要有输入数据
        ⋮
    ENDDO
```

程序节选

```
OPEN "Sales2.Dat" FOR INPUT AS #1
DateOfRun$ = DATE$
AreThereMoreRecords$ = "Yes"
GOSUB ReadModule
LET HoldDept = DeptNo
LET HoldSalesNo = SalesNo
GOSUB PrintHeadings
DO  WHILE AreThereMoreRecords$ = "Yes"
    GOSUB DetailProcessing
    GOSUB ReadModule
LOOP
```

在 DetailProcessing 中，将两个控制字段与其相应的保持域比较；如部门号和营业员号均无变化，只简单地将 Amtof Sale 加到 SalesTotal 上，并输入下一记录。因为该程序中不要求打印细目，所以在 DetailProcessing 中没有 LPRINT。

伪代码节选

```
DO  WHILE  只要有数据
    IF  部门号发生变化  THEN
        PERFORM
          :
          :    }   主控制中断过程
          :
        ENDPERFORM
    ENDIF
    IF  营业员号发生变化  THEN
        PERFORM
          :
          :    }   次级控制中断过程
          :
        ENDPERFORM
    ENDIF
    将 Amt 加到 Total
    PERFORM
        读记录
    ENDPERFORM
ENDDO
```

程序节选

```
DetailProcessing:
    IF  DeptNo <> HoldDept THEN
        GOSUB DeptBreak
    END  IF
    IF  SalesNo <> HoldSalesNo THEN
        GOSUB SalesBreak
    END  IF
    SalesTotal = SalesTotal + AmtOfSale
RETURN
```

当主结构定义后，将填充主级或部门变化过程，和次级或营业员变化过程。自顶向下程序首先开发所有逻辑骨架，细节留给以后开发。

简而言之，按自顶向下方式，首先编 DeptBreak 子程序，然后是 SalesBreak 子程序。部门号是主控制字段。

一个主要的中断控制程序应首先人工干预辅控制中断。也就是说，要做的第 1 件事是当部门号变化时处理前一部门的最后一个营业员的总计。在这里假设每个营业员为一个部门工作。因此，在 Dept Break 中的第 1 个指令是 GOSUB SalesBreak。这不仅打印前一营业员的总计，也增加营业员总计到部门总计，置 SalesTotal 初值为零，将新的 SalesNo 移到 HoldSalesNo 上。

在执行完 DeptBreak 子程序中的 SalesBreak 子程序后，需要完成下列步骤:

1. 打印DeptTotal。在PrintDeptTotalLine程序中做此工作
2. 置DeptTotal为零
3. 移DeptNo到HoldDept
4. 打印新页标题。

这样，DeptBreak 编码如下:

伪代码节选

```
IF  部门号发生变化   THEN
    PERFORM
      :
      :    }    以后填入部门号中断
      :
    ENDPERFORM
    PERFORM
        写 DeptNoTotal
    ENDPERFORM
    初始化 DeptNoTotal
    存 DeptNo
    PERFORM
      :
      :    }    以后再填入标题程序
      :
    ENDPERFROM
ENDIF
```

程序节选

```
DeptBreak:
    GOSUB SalesBreak
    GOSUB PrintDeptTotalLine
    DeptTotal = 0
    IF  AreThereMoreRecords$ = "Yes" THEN
        HoldDept = DeptNo
        GOSUB PrintHeadings
    END  IF
RETURN
```

当部门号变化时，叫主控制中断。注意到在测试辅控制中断之前先在 DetailProcessing 模块中测试主控制中断。作为结论，主级控制中断以人工干预次级控制中断为始。

DeptBreak 模块中最后的语句保证存入新的部门号和打印标题，除非所有记录已被处理过（AreThereMoreReconds$ 等于"NO"）。这个测试是必需的，因为 DeptBreak 子程序将在：

(1) DetailProcessing中执行，而后者在DO WHILE … LOOP中执行；

(2) 在EOF条件满足后，若所有记录已读入并处理过，必须进行人工中断。

请记住：自顶向下程序中，主程序比辅助程序先编码，从图7.9层次图可看出这些模型间关系。

即使部门号不变而营业员号发生变化，将导致辅助控制中断；即，SalesBreak。因此，营业员号与 HoldSalesNo 中值不等；或部门中断发生，将：

1. 打印前一营业员号的总计
2. 增加总计于DeptTotal
3. 置SalesTotal字段初值为零
4. 移新SalesNo到HoldSalesNo中

其执行过程如下：

伪代码节选

```
IF  营业员号发生变化   THEN
    PERFORM
        将 SalesNoTotal 移至输出域
            写记录
        将 SalesNoTotal 加到 DeptNoTotal
```

```
            初始化 SalesNoTotal
        存 SalesNo
    ENDPERFORM
ENDIF
```

程序节选

```
SalesBreak:
    GOSUB PrintSalesLine
    DeptTotal = DeptTotal + SalesTotal
    SalesTotal = 0
    HoldSalesNo = SalesNo
RETURN
PrintSalesLine:
    LPRINT TAB(30); HoldSalesNo
    LPRINT TAB(53);
    LPRINT USING "$#,###.##"; SalesTotal
RETURN
```

当所有记录读完并处理后，AreThereMoreRecords$ 将被置为"No"，而且 DO WHILE … LOOP 将结束，控制将返回循环后面的那个语句。对于单级控制中断处理，必须人工作出中断，以便打印出最后的 SalesNo 总计和最后的 DeptNo 总计。当 DO WHILE … LOOP 之后，执行主模块中的 DeptBreak；也就是说，当所有记录被读入并处理后执行。

请再注意，两级控制中断过程的所有伪代码和层次图均在图 7.8 和图 7.9 中分别说明，完整的 Turbo Basic 程序及输出见图 7.10。

记住程序可有许多控制字段，处理也如此，多级控制中断可干预次级控制中断。

§7.3 本章辅导

§7.3.1 本章小结

下面是一个程序框架，指出具有多个控制中断的处理过程：

```
GOSUB InitializeModule
GOSUB ReadModule
GOSUB PrintHeadings
MOVE control fields to hold areas
```

```
DO  WHILE AreThereMoreRecords$ = "Yes"
     IF  Control break THEN
         GOSUB ControlBreak
     ENDIF
     GOSUB DetailProcessing
     GOSUB ReadModule
LOOP
GOSUB ControlBreak
CLOSE
END
```

为完成控制中断处理，所有输入记录必须依主控制字段中的立即控制字段中的辅控制字段排序。如果记录不是依这种序，则文件必须在被处理之前依所要求的顺序排序。

§7.3.2　本章自测

1. 当每一个输入记录打印时，称之为＿＿＿＿打印。
2. 当成组输入记录时，归结成一行或多行打印，称之为＿＿＿＿打印。
3. （T or F）通常，打印单独记录比合计数据处理时间短。

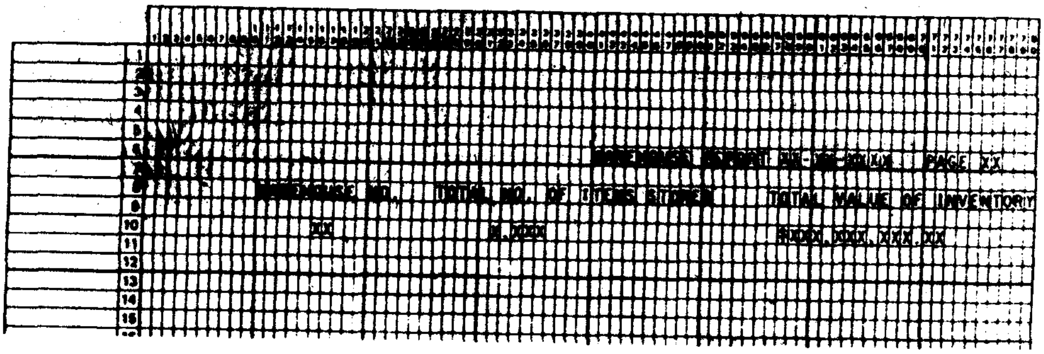
考虑如下问题描述。每一项储存一上输入记录，每个输入记录包含一个存有此项的库存数据，项数据和项所存的数值。

输入

WarehouseNo, ItemNo, QtyOnHand, UnitPrice, TotalValue

注：存储条目的总数等于各仓库的记录或条目(不是数量)之和。

输出



4. 如果输出报告包含每个库里货存总值，则称其为________报告.
5. 既然只打印库存信息，库存数据叫________字段.
6. 打印每个库货存总值的过程叫________过程.
7. 为用本章所描述的格式打印库存总计，输入数据必须按________排序
8. 假设数据排序正确，写出本问题的主模块.
9. 假定用户将细目模块将称作DetailProcessing，编写此模块.
10. 假定用户将控制中断模块称作ControlBreak，编写此模块.
11. 假定控制中断模块中有下列语句:

```
        ControlBreak:
            .
            .
            .
        IF   NOT EOF(1) THEN
             WHHold = WHNo
        END IF
        .
        .
        .
   RETURN
```

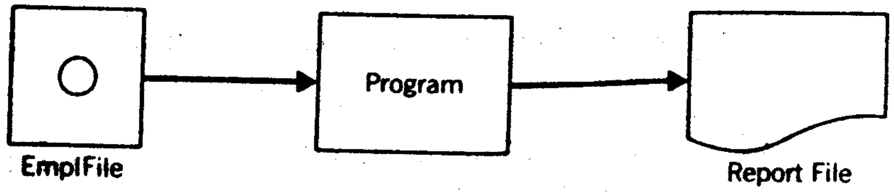
为什么在移 WHNO 到 WHHold 之前应检测文件结束条件?

12. 为避免上述比较，应使用什么技术?
13. 假定控制中断过程也需要总目录值的打印，指出得到最后总计的最佳方案.
14. 如程序中使用多级控制中断，产生多级控制中断的程序应从执行________开始.
15. (T or F) 当用两级控制中断时，输入数据必须依次字段中的主字段排序.

§7.3.3 实例

考虑下面问题描述:

系统流程图



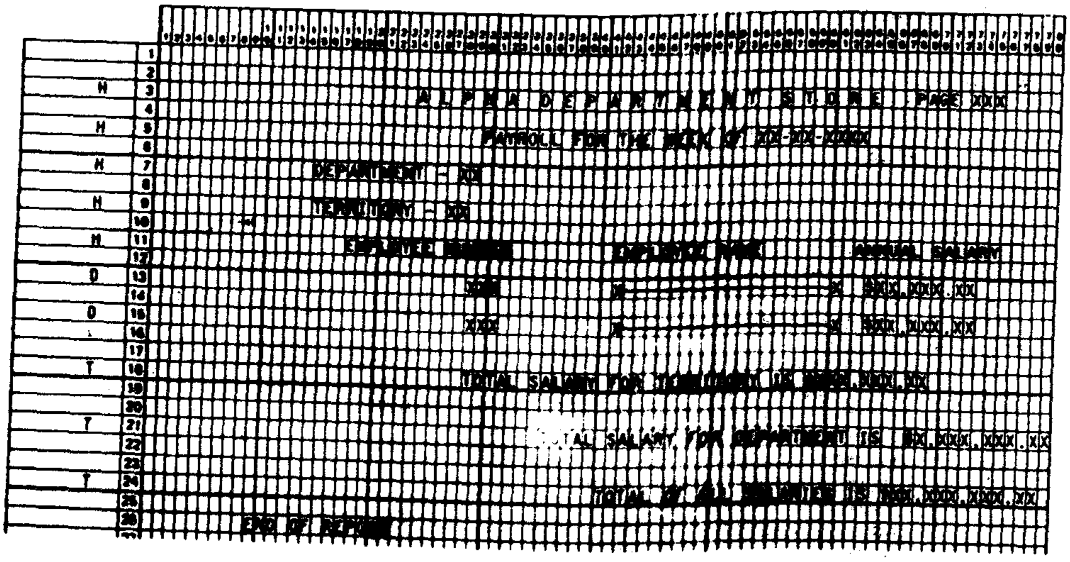
EmplFile 记录格式

DeptIn, TerrIn, EmpNo, EmpName$, AnnSal

Deptin 为主控制字段

TeerIn 为次控制字段

报表文件打印格式图



输入数据样例

```
01,01,001,"PAUL NEWMAN",31000
01,01,005,"ROBERT REDFORD",42000
01,02,007,"DIANA ROSS",41000
01,02,009,"TED KENNEDY",15000
02,07,023,"MICKEY MOUSE",27000
02,07,036,"DONALD DUCK",52000
03,009,054,"NANCY STERN",99999
```

写一个产生如上所述两级中断控制打印(图 7.11 给出解答和输出):

```
'Practice Program for Chapter 7
GOSUB InitializeModule
GOSUB ReadModule
LET HoldDept = DeptIn
LET HoldTerritoryNo = TerrIn
GOSUB PrintHeadings
'*****************************************
' The DO WHILE loop controls the overall
' processing
'*****************************************
```

```
DO WHILE AreThereMoreRecords$ = "Yes"
        GOSUB DetailProcessing
        GOSUB ReadModule
LOOP
GOSUB DeptBreak
GOSUB PrintFinalTotalLine
CLOSE #1
END
'*********************************************
' The Read Module reads input records
' and determines if an end of job has occurred
'*********************************************
ReadModule:
        IF NOT EOF(1) THEN
                INPUT #1, DeptIn, TerrIn, EmpNo, EmpName$, AnnSal
        ELSE
                LET AreThereMoreRecords$ = "No"
        END IF
RETURN
'*******************************************
' The Print headings module prints a heading
' at the beginning and for page overflow
'*******************************************
PrintHeadings:
        LPRINT CHR$(12)
        LET PageNo = PageNo + 1
        LPRINT TAB(24); "ALPHA DEPARTMENT STORE";
        LPRINT TAB(68); "PAGE";
        LPRINT TAB(73);
        LPRINT USING "###"; PageNo
        LPRINT
        LPRINT TAB(30); "PAYROLL FOR THE WEEK OF";
        LPRINT TAB(54); DateOfRun$
        LPRINT
        LPRINT TAB(15); "DEPARTMENT - ";
        LPRINT TAB(28);
        LPRINT USING "##"; DeptIn
        LPRINT
        LPRINT TAB(15); "TERRITORY - ";
        LPRINT TAB(28);
        LPRINT USING "##"; TerrIn
        LPRINT
        LPRINT TAB(18); "EMPLOYEE NUMBER";
        LPRINT TAB(41); "EMPLOYEE NAME";
        LPRINT TAB(63); "ANNUAL SALARY"
        LPRINT
RETURN


'*******************************************
' Detail Processing module processes each
' detail record
'*******************************************
DetailProcessing:
        IF DeptIn <> HoldDept THEN
                GOSUB DeptBreak
        END IF
        IF TerrIn <> HoldTerritoryNo THEN
                GOSUB TerritoryBreak
        END IF
        TerritoryTotal = TerritoryTotal + AnnSal
        GOSUB PrintDetailLine
RETURN


'*******************************************
' A change in Dept causes the Dept Break
' module to be executed
'*******************************************
DeptBreak:
        GOSUB TerritoryBreak
        GOSUB PrintDeptTotalLine
        FinalTotal = FinalTotal + DeptTotal
        DeptTotal = 0
```

```
        IF AreThereMoreRecords$ = "Yes" THEN
                HoldDept = DeptIn
                GOSUB PrintHeadings
        END IF
RETURN
'*****************************************************
' A change in Territory causes the Territory
' Break module to be executed
'*****************************************************
TerritoryBreak:
        GOSUB PrintTerritoryTotalLine
        DeptTotal = DeptTotal + TerritoryTotal
        TerritoryTotal = 0
        HoldTerritoryNo = TerrIn
        IF DeptIn = HoldDept AND AreThereMoreRecords$ = "Yes" THEN
                GOSUB PrintHeadings
        ELSE
                LPRINT
                LPRINT
        END IF
RETURN
'**************************************************
' The Print Detail Line module prints a
' detail line for each input record
'**************************************************
PrintDetailLine:
        LPRINT TAB(29); EmpNo;
        LPRINT TAB(42); EmpName$;
        LPRINT TAB(64);
        LPRINT USING "$$#,###.##"; AnnSal
        LPRINT
RETURN
'*****************************************************
' This module prints a territory total line
'*****************************************************
PrintTerritoryTotalLine:
        LPRINT
        LPRINT
        LPRINT TAB(28); "TOTAL SALARY FOR TERRITORY IS";
        LPRINT TAB(59);
        LPRINT USING "$$$#,###.##"; TerritoryTotal
RETURN
'*****************************************************
' This module prints a dept total line
'*****************************************************
PrintDeptTotalLine:
        LPRINT TAB(36); "TOTAL SALARY FOR DEPT IS ";
        LPRINT TAB(68);
        LPRINT USING "$#,###,###.##"; DeptTotal
        LPRINT
RETURN

'*****************************************************
' This module prints a final total line
'*****************************************************
PrintFinalTotalLine:
        LPRINT TAB(41); "TOTAL OF ALL SALARIES IS";
        LPRINT TAB(66);
        LPRINT USING "$$#,###,###.##"; FinalTotal
        LPRINT
RETURN
'***************************************************
' This module opens the input file and
' initializes variables
'***************************************************
InitializeModule:
        OPEN "Payroll.Dat" FOR INPUT AS #1
        DateOfRun$ = DATE$
        AreThereMoreRecords$ = "Yes"
RETURN
```

图 7.11(a) 实例的解答

ALPHA DEPARTMENT STORE PAGE 1

PAYROLL FOR THE WEEK OF 02-15-1989

DEPARTMENT - 1

TERRITORY - 1

EMPLOYEE NUMBER	EMPLOYEE NAME	ANNUAL SALARY
1	PAUL NEWMAN	$31,000.00
5	ROBERT REDFORD	$42,000.00

TOTAL SALARY FOR TERRITORY IS $73,000.00

ALPHA DEPARTMENT STORE PAGE 2

PAYROLL FOR THE WEEK OF 02-15-1989

DEPARTMENT - 1

TERRITORY - 2

EMPLOYEE NUMBER	EMPLOYEE NAME	ANNUAL SALARY
7	DIANA ROSS	$41,000.00
9	TED KENNEDY	$15,000.00

TOTAL SALARY FOR TERRITORY IS $56,000.00

TOTAL SALARY FOR DEPT IS $ 129,000.00

ALPHA DEPARTMENT STORE PAGE 3

PAYROLL FOR THE WEEK OF 02-15-1989

DEPARTMENT - 2

TERRITORY - 7

EMPLOYEE NUMBER	EMPLOYEE NAME	ANNUAL SALARY
23	MICKEY MOUSE	$27,000.00
36	DONALD DUCK	$52,000.00

TOTAL SALARY FOR TERRITORY IS $79,000.00

TOTAL SALARY FOR DEPT IS $ 79,000.00

ALPHA DEPARTMENT STORE PAGE 4

PAYROLL FOR THE WEEK OF 02-15-1989

DEPARTMENT - 3

TERRITORY - 9

```
EMPLOYEE NUMBER        EMPLOYEE NAME        ANNUAL SALARY

      54               NANCY STERN          $99,999.00

        TOTAL SALARY FOR TERRITORY IS   $99,999.00

             TOTAL SALARY FOR DEPT IS        $   99,999.00

                 TOTAL OF ALL SALARIES IS    $307,999.00
```

图 7.11(b) 输出样例

§7.3.4 复习题

True−False(是非)题

1. 为用中断控制过程，输入数据必须依控制字段排序
2. 中断控制过程主模块中，在读第1个记录之前已完成标题程序
3. 在控制中断过程和主模块中细目或计算程序执行之前，控制字段必须先移入保持域
4. 当所有数据被读入，处理且控制返回LOOP后的语句，其它处理只剩当需要时打印累计最终小计，然后用户可 CLOSE 和 END
5. 在细目处理模块，可先测试次级控制中断，然后测试主级控制中断
6. 细目模块经常打印输出
7. 在中断控制模块中减少错误的方法之一是在检查控制字段是否排序正确时执行序列检查程序
8. 控制中断程序必须经常将总计字段清零
9. 中断控制模块通常包含WRITE语句
10. 在控制中断程序中允许两级控制中断的最大值

§7.3.5 调试练习

```
CLS
OPEN "Transaction.Dat" FOR INPUT AS #1
GOSUB InitializeModule
LET HoldAcct = AcctNoIn
DO  WHILE AreThereMoreRecords$ = "Yes"
    IF  AcctNoIn <> HoldAcct THEN
           GOSUB AcctBreak
```

```
    ELSE
            Total = Total + Amt
            LPRINT TAB(20); AcctNoIn;
            LPRINT TAB(35); Amt
END
InitializeModule:
    DateOfRun$ = Date$
    AreThereMoreRecords = "Yes"
    GOSUB ReadModule
RETURN
ReadModule:
    IF  NOT EOF(1) THEN
            INPUT #1, AcctNoIn, Amt
    ELSE
            LET AreThereMoreRecords = "No"
    END IF
RETURN
AcctBreak:
    LPRINT TAB(15); "TOTAL AMOUNT FOR ACCOUNT IS";
    LPRINT TAB(40);
    LPRINT USING "$$#,###.##"; Total
    LPRINT
    .
    .
    .
RETURN
```

---

1. 这个逻辑与正文中的例子不同，这个逻辑结构对吗？
2. 有两个语句在AcctBreak中丢失，这将导致逻辑致,为什么？
3. 在DO WHILE … LOOP完成后，应该用GOSUB去打印最后控制块吗？请作出解释。
4. 假设忽略了主程序中的:

   LET HoldAccct = AccctNo

   语句。其中有没有替代处理？并作出解释。
5. 这个程序不产生标题，修改原码使插入正确报告和栏目标题。

§7.3.6 编程题:

1. 写一个程序，打印磁盘记录中每五天的销售总计，问题描述见图7.12

   注:

   a. 营业员每一次销售均有磁盘记录；这样数据是无穷无尽的。

   b. 记录依日数据排序，从星期一到星期五。

SalesTrans 的记录格式

DayNo, SalesPerson, SalesAmt

销售报表打印格式图

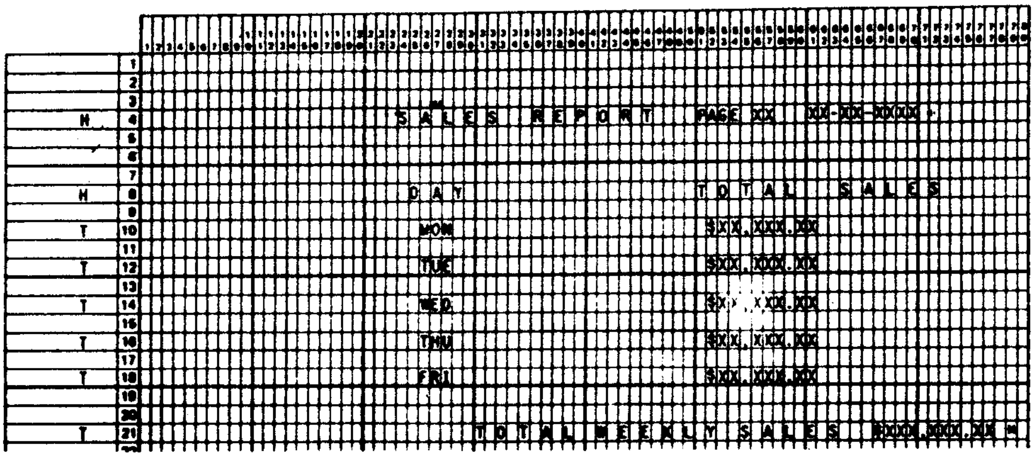


图 7.12　编程题 1 的问题描述

2. 编写程序按范围打印职员薪水总计，输出格式见图7.13.

注:

a. 输入记录依范围排序，使之具有下面格式:

EmpNo，EmpName$，TerrNo，Salary

b. 打印每个范围的总薪水，在报告尾部，打印整个公司总薪水.

c. 有三个范围：01，02，和03.

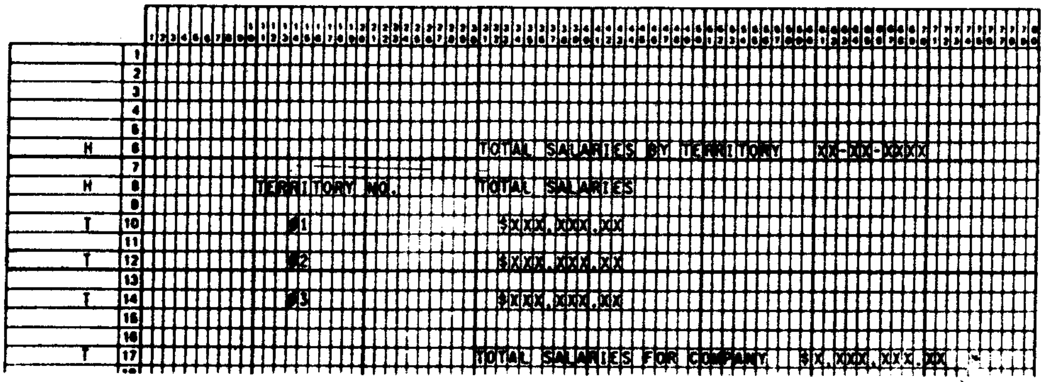
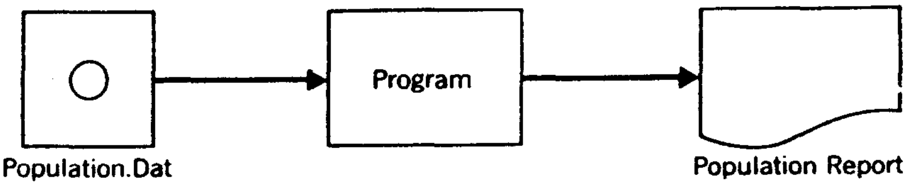


图 7.13　编程题 2 的输出格式

3. 编写程序打印每个州总人口，问题描述见图7.14.

注：记录依州中的县排序

系统流程图



Population.Dat 的记录格式

StateNo, CountyNo, DistrictNo, Population

人口报表打印格式图

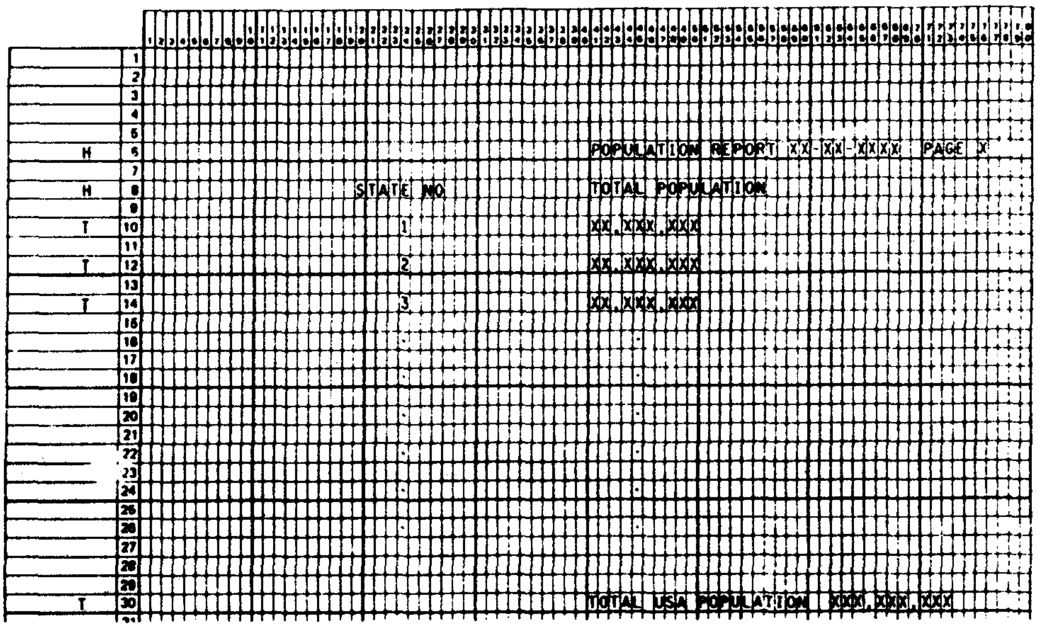


图 7.14　编程题 3 的问题描述

4. 编写程序绘制地区、部门范围内的职号数目表，输入磁盘，记录如下格式:

TerrNo, AreaNo, DeptNo, EmpName$, Salary

注:

a. 有3个范围，每个地域范围有3个区，每个区10个部门.

b. 磁盘文件依地区范围内的部门排序.

c. 输出报告格式如图7.15.

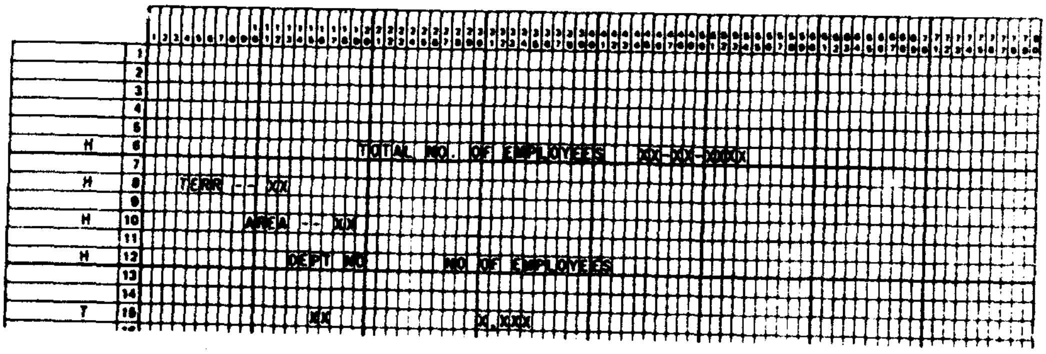


图 7.15　编程题 4 的输出格式

第四部分
程 序 调 试

本部分内容

第八章　调试程序、确认数据和改善程序设计
第九章　深入了解 Turbo Basic 环境

第八章　调试程序、确认数据和改善程序设计

本章要求：

1. 了解可能发生的各类输入错误.
2. 掌握确认输入数据的方法.
3. 掌握发现输入错误时应采取的措施.
4. 掌握调试方法.
5. 熟悉READ 和DATA语句.

§8.1　程序错误

每个人都可能听说过有关"计算机错误"的可怕故事. 报纸上几乎每天都有关于人们错误地接到数目荒谬的计算机帐单或支票的报道. 这种错误的发生，在大多数情况下，是由于程序错误或输入错误，而不是计算机功能失灵. 本节重点介绍查找和减少程序错误的方法. 下一节重点介绍查找和减少输入错误的方法.

有两种类型的程序错误——语法错误和逻辑错误. 下面详细介绍各类错误.

§8.1.1　语法错误

正如我们在本书中始终强调的一样，设计良好的程序应先规划、后编码，并在实际键入之前作最后的"桌面检查"或复查. 但是，许多程序员急于开始，坐在键盘旁便开始程序设计过程，并直接键入程序. 不幸的是，这种程序往往规划不充分，缺乏正确的总体结构或设计.

我们极力鼓励用户在编程之前仔细地对程序作出规划. 在程序编写完毕并输入计算机之后，应仔细检查录入错误. 此外，还应再次复查程序的逻辑结构，以便发现错误.

从头到尾地进行桌面检查之后，准备编译程序或将它翻译成机器代码. 在翻译过程中，编译器将显示可能发生的任何违反程序设计规则的错误. 这种违反程序设计规则的错误叫"语法错误（syntax errors）". 这些错误必须在程序可运行之前修正.

记住，为了减少运行程序时所需的击键数目，在 Turbo Basic 中使用 Run 命令同时编译和运行程序. 即，在程序存盘之后，先加亮主选单中的 Run 命令. 然后再按 Enter 键（或者用另一种方法，在主屏幕中按 R）. 不过，请注意，如果有任何语法错误，程序将不能实际运行. 语法错误必须在程序可运行之前修正.

不用 Run 命令立即执行程序时，可以先编译（在主屏幕上按 C 或加亮 compile 并按

Enter)，后运行。在编译过程中，若有语法错误，则将其列出．在程序可实际运行或执行之前，必须将它们修正。第九章详细介绍 Compile（编译）命令．

用 Compile 或 Run 命令编译程序时，Tubo Basic 将以下述方法指出语法错误．

1. 在Edit窗口顶部打印错误码，并附有语法错误的简要描述信息．
2. 指出发生错误的程序元素：即光标在程序中发现错误的位置闪烁．

如果发现语法错误，编译器自动回到 Edit 模式下，以便于用户可立即修正．Turbo Basic 将语法错误称作"编译错误（compiler error）"。

§8.1.1.1 语法错误信息

语法或编译错误信息按下述格式在 Edit 窗口顶部显示：

Error 错误号： 错误描述

图 8.1 和图 8.2 给出了语法错误信息的两个例子：

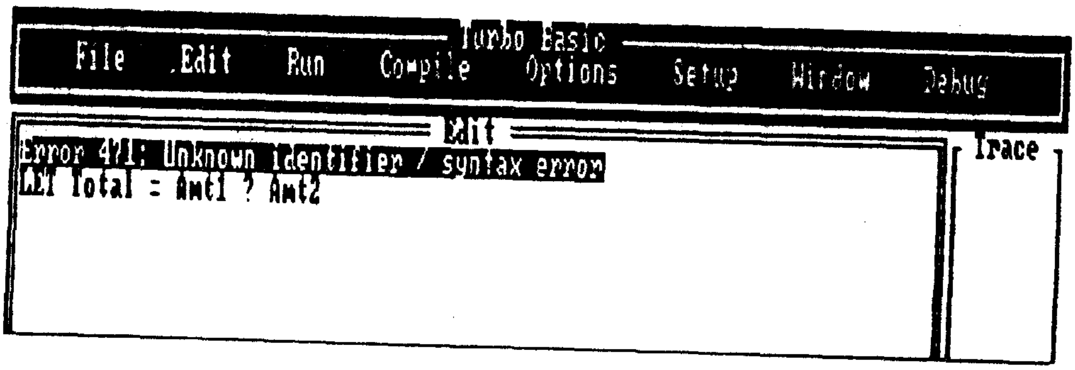


图 8.1 语法错误信息实例（一）

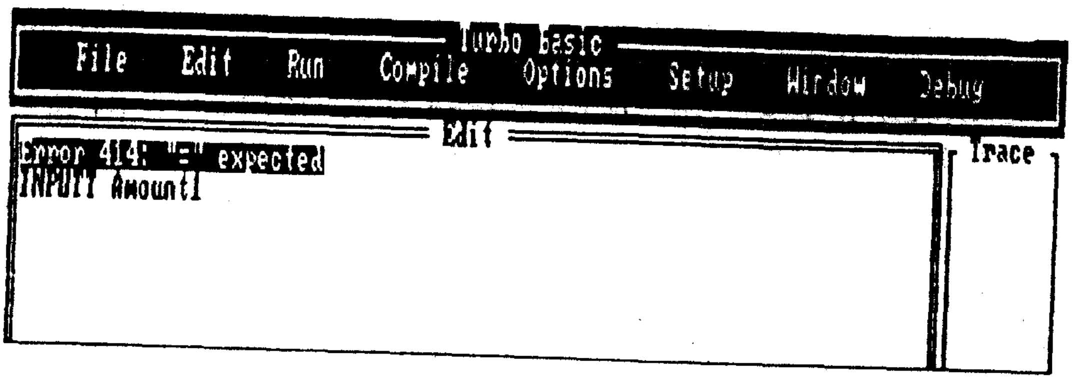


图 8.2 语法错误信息实例（二）

错误信息有时是不言而喻的，而有时可能难于确定导致错误的原因。《Turbo Basic 用户手册》中含有各错误的简要解释；用户手头若有这本手册，可能会有所帮助（为方便读者，本书附录部分给出 Turbo Basic 的错误信息，并附有简明扼要的解释）。注意，语法错误信息中错误号至少为 256。

当程序元素明显错了时，错误信息将是一目了然的。例如，在图 8.1 的例子中，“ ? ”显然错了，应用算术运算符代替。

在其它情况下，如上图 8.2 中的例子，错误信息就不那么明了；用户可能立即意识到“INPUTT”拼写错了，但奇怪为什么应该有一个等号(“ = ”expected)。如果编译这条指令时，用户便可注意到光标指向变量 Amount1，而不是 INPUTT，可能引起混乱。

因为 INPUTT 不是 Turbo Basic 的保留字，所以编译器将它假定为一个变量名。当变量名为一行中的第一个元素时，编译器便假定该行应包括一条赋值语句，而赋值语句由变量名（或任选词 LET）打头，后随一个等号。这就是为什么图 8.2 中的例子要求在 INPUTT 后面加一个等号的原因。即使错误信息不那么显而易见，但随着编译经验的积累，用户很快便能识别常见的错误。

总之，大多数错误信息还是易于理解的。另一些错误信息在弄清楚其含义之前要求有一定的经验和／或说明。编写和调试的程序越多，对 Turbo Basic 错误信息的熟悉程度也就越深。

在此，不可能介绍所有的语法错误，下面仅给出几个经常碰到的错误信息。

§8.1.1.2　语法错误举例

下面给出几个常见的错误信息，并作出解释：

412　“(” expected
　　该语句的语法格式要求有一个左括号。

413　“)” expected
　　该语句的语法格式要求有一个右括号。

418　Numeric expression reguires relational operator
　　编译器在应该是数字运算符的地方发现字符串操作数。

432　IF expected
　　编译器发现一个END IF或 EXIT IF 语句，而未见开头的IF 语句。

435　DO Loop expected
　　编译器发现一个LOOP 或EXIT LOOP 语句，而未找到开头的DO语句。

438　FOR LOOP expecteo
　　编译器发现一个 EXIT FOR 语句，而未找到开头的 FOR 语句。

442　LOOP ／WEND expected
　　DO 或 WHILE 循环没有用相应的 LOOP 或 WEND 语句中止。

445 NEXT expected

FOR 循环未用 NEXT 语句正确中止。

446 THEN expected

IF 语句丢失它的伴随部分 THEN

其它常见的语法错误在本节后面部分介绍。

编译程序时，只要发现语法错误，便中止编译过程。如前所述，编译器同时也将该程序返回到 Edit 模式下，以便用户可立即修正。各语法错误必须修正，并再次编译新版本的程序。在随后的编译过程中，如果发现其它语法错误，编译器同样一次一个地将它指出。在编译或运行之前，最好先将程序存盘：在 Edit 模式下按功能键 F2 或选用 File 命令下的 Save 选项。

由于程序中，尤其是大的、复杂的程序，常含有许多语法错误，因此可能需要重复进行几次编译、修正、存盘和再编译等过程。如果确实如此，不要失去信心。这种重复过程在程序设计中是常见的。不过，请注意，如果花点时间将程序仔细规划一下，进行桌面检查，并在键入后再次检查的话，那么将发生的语法错误数目往往可减少到最小程度。

§8.1.2 逻辑错误

正如我们已经看到的，语法错误相对容易找出和修正。计算机在所有语法错误全部修正之前不运行相应的程序。

程序在完全编译之后，如果未发现语法错误，那么它便可运行测试数据（test data），以确保无逻辑错误（logic error）。如前所述，有两种方法，可用来编译和运行程序。可以先选 Compile 命令；如果编译时未发现语法错误。然后再选用 Run 命令测试程序的逻辑。或者，刚开始就选 Run 命令；在这种情况下，如果没有语法错误，程序将在编译之后自动执行。

逻辑错误可能导致：

(1) 程序中断或运行时错误（Run-time error）；

(2) 错误的输出；

(3) 死循环。

如果发生运行时错误，将给出一条错误信息。在错误输出或死循环的情况下，用户必须自己找出错误。对于一个程序来说，所有的逻辑错误均被改正才算充分调试。

记住：运行程序时，逻辑错误可能源于后续指令的错误，也可能源于一条未完成所期望功能的、被错误编写的指令。

逻辑错误通过先返回 Edit 模式，查出错误指令，然后再编译和重新执行的方法修正。下面详细讨论各种逻辑错误。

§ 8.1.2.1 运行时错误

考虑如下程序:

```
INPUT Amt1
LET Price = Amt1 / B
PRINT Price
```

因为所有指令的语法格式全都正确，所以上述程序不会发生任何语法错误．但是，运行时，正如所显示的，因为 B 的值未给定，它被赋以 0，所以将导致运行时错误．运行时错误象编译错误一样，有一条显示在 Edit 窗口顶部的错误信息．在上述情况下，在运行过程中,将给出如下错误信息:

```
Error 11: Division by zero.
```

同样,《 Turbo Basic 用户手册 》也提供了所有运行时错误的简要解释（本书附录部分有此内容）;运行时错误的错误号为: 1～255.

常见的运行时错误信息有:

11　Division by zero
　　试图除零，或计算 0 的负指数．

13　Type mismatch
　　在应该用数字数值的地方用了字符串数值；反之亦然．

25　Device fault
　　发生硬件错误

27　Out of paper
　　打印机接口指出打印机发生故障．打印机可能被关、缺纸、或其它问题．

象发生编译错误时一样，发生运行时错误时，编译器将光标指向引发错误的程序条目，并自动返回 Edit 模式下，以便于用户修正．此后，该程序必须重新编译和重新运行．

本书后面部分介绍其它常见的运行时错误．

§ 8.1.2.2 错误输出

一个程序在编译时可能无任何错误，而且在运行时也可能无任何错误，但检查其结果时，程序员发现它是错的．程序的逻辑结构不正确，指令顺序错误，操作不正确等，都可

能引发这种情况.

因此，导致错误输出的逻辑错误在检查计算机产生的结果时发现. 找出所有这种逻辑错误的唯一的办法是：确保测试数据的完备性，并且仔细检查程序的输出.

有时，可导致错误输出的逻辑错误要到定期操作或运行之后才能发现. 这种情况的发生是因为：只有输入一个不常用的输入数据时，这种逻辑错误才会变得明显. 在程序已被定期使用之后，修改这种错误是非常困难的. 通常费用也大. 程序员可能不再熟悉程序的所有方面. 结果是，他（她）可能作出了混有原先问题的修改或引入其它问题. 此外,定期使用该程序的公司在修正这种错误期间往往要赶上业务积压.

§ 8.1.2.3 死循环

如果程序在运行状态下输出正确，但计算机未显示任何信息，看起来好象已停止工作，那么可能产生了死循环. 同样，如果产生的输出由重复显示的相同的数据所组成，那么它往往是由死循环而引起的错误结果. 为了在这种情况下中止程序的运行，按 Ctrl-Break 键（按 Ctrl 键，并在按 Ctrl 键的同时按 Break 键）. 如前所述，Break 键通常位于键盘右上部，也可能标作 Scroll Lock 或 Pause . 据第二章所述，只有 Options 命令下的 keyboard break 选项设为 ON 时，Ctrl -Break 才能有效地中止程序的运行. 一旦中断了含有死循环的程序，就必须复查程序的逻辑结构，看一看能否找出错误所在.

总之，在所有语法错误均修正之后，必须仔细检查程序的逻辑错误.

下述测试数据的完备准备原则在调试阶段将有助于逻辑错误的发现.

§ 8.1.3 使用全面且现实的测试数据将减少未发现的逻辑错误

必须用测试数据进行测试，以确保它可正确地运行于所有情况下. 由于调试工作取决于完备而设计良好的测试数据，因而测试数据的准备是程序员的重要责任. 测试数据不完备，程序将不能完全调试，逻辑错误或错误输出在日后、在定期运行过程中出现的可能性就存在.

测试数据应包括程序测试的每一个状态；至少应包括一个为各给定状态设定的变量. 此外，为确保多个目的的设计程序每次均能实际地正确运行，应提供充足的测试数据. 例如，一个模块第一次可正确地运行,但重新设置某些变量后不能运行. 这就意味着，特定的错误只有在第二次运行时才可能发现. 因此，为保证这种模块可多次运行，必须有足够的测试数据.

例如，下述程序包含一个写完 25 行才执行的标题子程序.

```
IF  LineCounter  >  25  THEN
    GOSUB PrintHeading
END  IF
```

如果 PrintHeadings 在 LineCounter 为 0 时不能运行，那么这个错误在第一次运行

PrintHeadings 时将发现不了，只有再次运行才能发现。

下面是测试程序时应遵守的原则:

1. 保证测试数据包含所有可能的数值.

   测试数据既要包含有效数据，也要包含无效数据，以确保程序可检查无效数据。后面，我们将给出一个用于测试无效数据的子程序例子。

   应用足够的输入数据生成有效的输出；例如，如果输出通常包含许多页，那么在测试过程中应至少生成两页，以检查行数的重新设置，页数的正确增加，等等。

2. 如有可能，用"真实的"或"现场的"的数据测试程序

   若有可能，用"真实的"或"现场的"输入作为测试数据，或者让其它人准备测试数据。这通常不失为一个好主意。因为，编写程序的程序员可能看不到其中的一些明显的疏忽之处。如果程序有遗漏之处，那么测试数据可能也有类似的遗漏。例如，程序员若忘记加入给定条件下的错误处理单元，那么他(她)同样有可能忘记加入测试该特定错误的的测试数据。使用"真实的"戒"现场的"输入，或者让其它人准备测试数据可减少这种差错。

3. 用测试数据进行结构走查.

   测试数据准备好之后，进行结构走查，以确定程序正常运行时计算机应产生的结果。这就意味着，用户应人工逐步通过程序的逻辑结构，以确定计算机应产生的输出。然后，用测试数据运行程序，并将计算机产生的结果和走查的结果进行比较。如果结构走查产生的结果与计算机的相同，那么该程序是正确的，并被认为是调试好的。否则，必须找出错误的根源，并将它修正。然后再重新编译和重新运行该程序.

§8.1.4 修正逻辑错误

有时，逻辑错误的发生难于发现。调试方法可用来检查逻辑结构和帮助识别逻辑错误。出现一个程序员不能简单地找出的错误时，这种方法特别有用，不仅如此，这种方法还可用来检查看似运行正常的程序。即，它们有助于找出其它方法发现不了的逻辑错误。

在 Turbo Basic 中，有三种主要的调试方法用于逻辑错误的查找:

调试方法

1. 跟踪 (Trace)。
2. 调试其它段时,跳过已调试的段。
3. 在确定的检查点打印中间结果。

§8.1.4.1 使用 Trace 选项

Turbo Basic 程序中的逻辑可以跟踪；即，在 Trace 窗口中显示实际执行的各条指令的行号．这样，准确地看到计算机如何一步一步地运行程序．

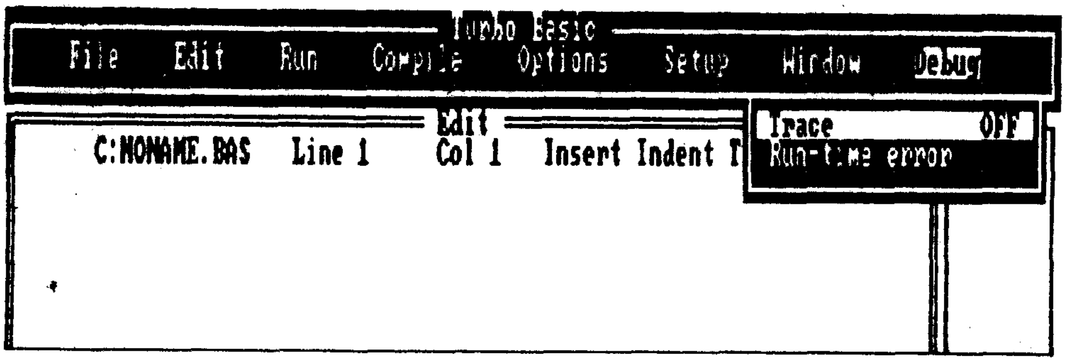
为有效地使用 Trace 选项调试程序,必须使用行号．Trace 窗口将显示当前执行指令的行号．如果在 GOSUB 语句中使用了标号，执行 GOSUB 命令时，标号也将出现在 Trace 窗口中．

有如下几种方法可用来跟踪程序:

§8.1.4.1.1 选用 Debug 命令下的 Trace ON 选项

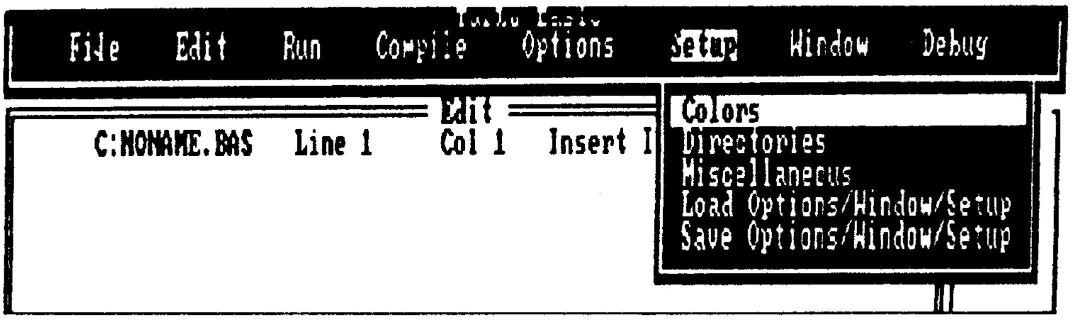
如第二章所述，使用 Debug 命令下的 Trace ON 选项，可跟踪整个程序中的逻辑结构．为此，在主选单中用光标键加亮 Debug 命令，并按 Enter 键；另一种方法为，在主选单中简单地按字母 D 即可．

此时，下述下拉选单将出现:

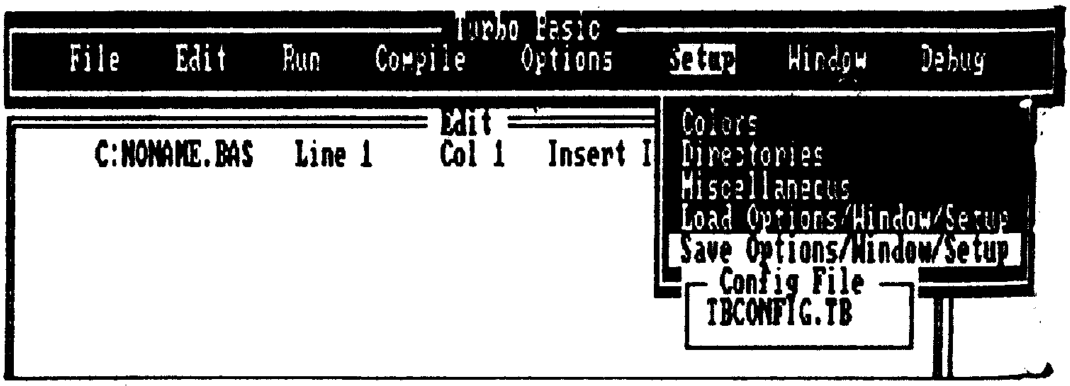


其中，Trace OFF 行被加亮，作为缺省设置．为了将 Trace OFF 改为 Trace ON，仅需简单地按一下 Enter 键即可．Enter 键的功能类似于一个乒乓开关: 若按一次 Enter，Trace OFF 则变为 Trace ON，如果按两次 Enter，则 Trace ON 再变为 Trace OFF.

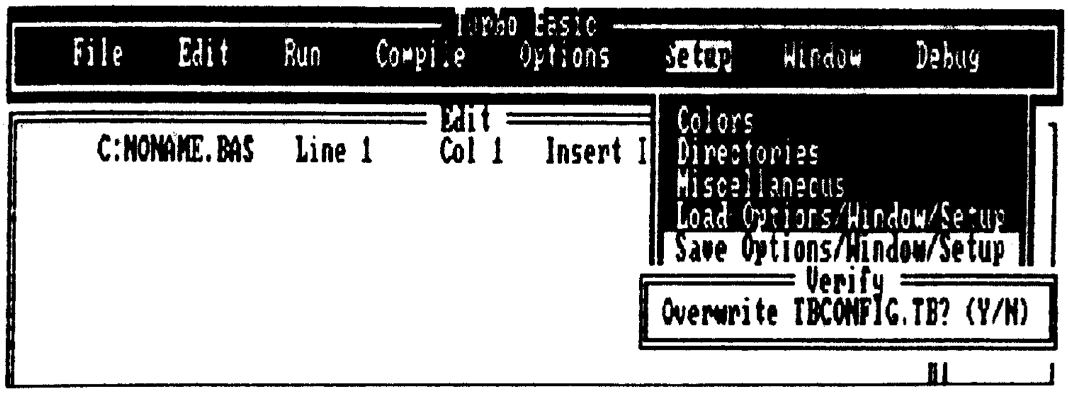
设定 Trace ON 时，便可跟踪未正确工作的单独程序．或者，将 Trace ON 选项存入配置文件 TBCONFIG.TB，以便于缺省跟踪所有程序．为此，选设定 Trace ON 选项，然后再按 Esc 键退回主选单；用光标键加亮 Setup 命令，并按 Enter．此时，出现下述下拉选单:



接着，按四次光标键↓加亮 Save Options / Window / Setup 选项，并按 Enter。随即一个上托选单出现在下拉选单之上:



如果按 Enter 键，Turbo Basic 便询问是否确认要修改标准的配置文件:



若按 Y，Trace ON 便成为配置文件的一部分，并在每次启动 Turbo Basic 时缺省设定。Trace On 为缺省设置时，还可以为任何指定的程序运行关闭跟踪功能。方法是：加亮 Debug 命令，并选定 Trace OFF。

总之，首次使用 Tubo Basic 系统时，Trace OFF 为缺省设置。用上述方法可将缺省设置改为 Trace ON。Trace ON 既可为单独的程序设置，也可以设定为缺省状态。

Alt–F9 和 Alt–F10 用于程序跟踪

Trace 选项为 ON 时，Trace 窗口将在运行过程中列出所执行语句的行号和标号。开

始运行时，程序中指定的第一个行号将打印在 Trace 窗口中。例如：

```
┌──── Trace ────┐
│ [10]          │
└───────────────┘
```

必须按 Alt–F9 才真正开始运行程序；即在键盘上按 Alt 键，并在按住 Alt 键的同时按功能键 F9（Alt 键位于标准 PC 键盘的左边，或在扩充键盘的右下部）。在运行过程中，若再次按 Alt–F9，将停止程序的运行。因此，Alt–F9 控制程序运行的开始和中止。

按 Alt–F10 键，可单步通过程序的逻辑结构，一次一行。每按一次 Alt–F10，Trace 窗口将显示下一条被执行的指令的行号。为了回到自动运行状态以便于程序的正常运行，不必在每条指令上停止，可再次按 Alt–F9。

注意：在运行状态下，屏幕底部的帮助行给出在此所介绍组合键的意义：

```
Alt–F9     Stop / Continue
Alt–F10    Step
```

Alt–F9 键的功能也好像一个乒乓开关。为开始运行，按一次 Alt–F9，而按两次 Alt–F9 便中止程序的运行。

§8.1.4.1.2　在程序中使用 TRON 指令

作为设置 Trace ON 选项的另一种方法，位于 Turbo Basic 程序起始处的 TRON 指令也将列出运行过程中所执行程序的行号。为使用 TRON 指令、不必将 Debug 命令下的 Trace 选项设为 ON。

考虑如下程序：

```
10  CLS
20  PRINT "Begin Execution"
30  PRINT Amt1
```

假设 Trace 选项为 ON，运行这个程序时，必须按 Alt–F0 键开始执行；此时，Trace 窗口中出现如下信息：

```
[10] [20] [30]
```

由此可知，行号为 10，20 和 30 的语句依次执行，然后程序停止。

例 1　考虑如下程序，并假定Trace ON 选项来选定：

```
10  TRON                    'Sets the Trace option ON
20  INPUT Amt1, Amt2
30  DO  WHILE Amt1 < 999
40      LET Total = Amt1 + Amt2
50  LOOP
60  PRINT Total
70  INPUT Amt1, Amt2
    .
    .
    .
```

在运行过程中，[20]首先出现在 Trace 窗口中；此时，必须按 Alt−F9 才能继续运行．接着，程序提示输入数据．为 Amtl 键入一个数值后，假若[30][40][50][80]出现在 Trace 窗口中，因为行号为 40 的语句被执行，所以计算机确定输入的 Amtl 在 15～25 之间．

例 2 考虑如下程序：

```
10  TRON                'Sets the Trace option ON
20  INPUT Amt
30  IF  Amt > 15 AND Amt < 25 THEN
40      LET Total = Total + 1
50  ELSE
60      LET Total2 = Total2 + 1
70  END IF
80  PRINT ...
    .
    .
    .
```

该程序中有一个逻辑错误．如果初始的 Amtl 小于 999，则执行 DO WHILE 结构，而 DO WHILE...LOOP 只简单地重复增加 Total．由于 60 和 70 行的 PRINT 和 INPUT 指令未在该循环体内，因此，它们永远也不会被执行．结果是，因为 Amtl 永远不会变化，故循环体将无限次地重复执行．由于该程序中使用了 TRON 指令，所以[20]在刚开始便出现在 Trace 窗口中．按 Alt−F9 键之后，程序提示输入数据；输入数据时，下列行号将显示在 Trace 窗口中．

[20][30][40][50][40][50][40][50]...

40 和 50 行的重复执行表示出现了死循环．为中止这个死循环，按 Alt−F9．为了一次一条指令地单步通过程序逻辑，可以按 Alt−F10，显示下一条被执行指令的行号，并停止．每按一次 Alt−F10 键，Trace 窗口中便显示下一条被执行指令的行号．如果在上述程序中重复按 Alt−F10，那么 Trace 窗口中将显示：

[40][50][40][50]...

Alt–F10 键的使用有助于单步通过程序的逻辑结构，以便更清楚地看到指令执行的顺序.

用 TROFF 指令关闭 Trace 选项

使用 Trace ON 选项，而不是 TRON 指令时，整个程序将被跟踪. 不过，TRON 指令不必放在程序的开头，可用来跟踪程序的一部分. 也就是说，TRON 可在程序中任何想跟踪程序逻辑的地方编写. 同样，此后可以用 TROFF 指令完全关闭跟踪功能. 假设出现了一个逻辑错误，并知道程序中 120 至 250 行之间有一个错误. 为了跟 120–250 行间的逻辑结构，编程如下:

```
10
 .
 .
 .
115   TRON
120                          'Portion of program that
 .                           'will be traced
 .
 .
250
255   TROFF
 .
 .
 .
END
```

> **编程指南**
>
> 有些程序员在调试阶段总是跟踪整个程序. 这样可单步通过每次运行的逻辑. 为此，Debug 命令下的 Trace ON 选项应存入配置文件 TBCONFIG.TB；或者，在所有程序的开头加入 TRON 指令.
>
> 而另一些程序员只在发生他们不能识别的错误时使用 Trace 选项；这些程序员喜欢有选择地使用 TRON 和 TROFF 指令跟踪部分程序.
>
> 建议用户采用最适合于自己的方法.

请记住，为跟踪程序或任意程序段，必须使用行号. 若只跟踪程序的一部分，则只有这部分需要有行号:

```
        INPUT ...
        LET ...
        .
        .
        .
        TRON
110     .            } Only the instructions to be
        .            } traced need line numbers
250     .
        TROFF
        PRINT ...
        .
        .
        .
END
```

在任一种情况下，一旦所有的逻辑错误均已修正，并程序完全调试过了，则应删除程序中含有的 TRON 和 TROFF 语句。同样，应为定期使用的程序关闭 Trace 选项。不应让完全调试好的程序在定期运行时打印它碰到的每个行号。

还应注意，GOSUB 语句中引用的标号也在 Trace 窗口中列出。例如:

```
10   INPUT ...
20   GOSUB XYZ
30   PRINT
40   END
XYZ: ...
     .
     .
     .
RETURN
```

如果选定 Trace ON 选项，那么 Trace 窗口中将出现下述信息。

[10] [20] [XYZ] [30]

关于行号的警告

注意，其它版本的 Basic，要求有行号，而不象在 Turbo Basic 中可有可无。在这些版本中，行号实际上确定指令执行的顺序。

考虑如下程序:

```
10   INPUT Amount
30   PRINT Amount
20   LET Amount = Amount + 10
```

在其它版本的 Basic 中，计算机将按行号顺序（[10][20][30]）执行指令；即使按不同的顺序键入也如此。

假设为 Amount 输入一个 5，使用其它版本的 Basic 时，Amount 将打印为 15。这种情况的出现是因为 20 行在 30 行之前执行。也就是说，在其它版本的 Basic 中，是行号，而不是指令编写的顺序决定执行的顺序。

而在 Turbo Basic 中，不管使用的行号如何，指令执行的顺序总是它们在屏幕上显示的顺序。因此，行号只对程序员有用，而对指令执行的顺序毫无影响。于是，在 Turbo Basic 中，上述程序将打印 Amount 的值为 5，而不是 15。

还应注意，其它版本的 Basic 用 LIST 和 LLIST 按行号顺序，而不是实际录入顺序显示或打印程序清单。因为 Turbo Basic 严格地按屏幕显示顺序编译和运行程序，根本不管可有可无的行号，因此不需要 LIST 和 LLIST 命令。于是，这些命令不可用于 Turbo Basic 之中。

§8.1.4.2 调试其它段时跳过已调试的程序段

在大型程序中，可能有许多程序段或模块工作正常，因而在后面的测试运行过程中，不必测试。在这种情况下，用单引号“ ' ”标记这些代码行，以便编译器将它们当作注释处理。用这种方法，可避免在进一步的测试过程中执行它们。

考虑如下程序:

```
.
.
.
IF  LineCt    25 THEN
    GOSUB PrintHeadings
END IF
.
.
.
END
PrintHeadings:
    LPRINT
    .
    .
    .
RETURN
```

假设我们知道 PrintHeadings 模块工作正常，并且不想在剩下的测试过程产生标题，那么将 IF-THEN-END IF 条件块标记为注释。

```
'IF   LineCt > 25 THEN
'     GOSUB PrintHeadings
'END  IF
```

或者将 PrintHeading$ 模块中的指令全部标作注释:

```
PrintHeadings:
'     LPRINT
'       .
'       .
'       .
RETURN
```

在两种情况下，程序完全调试之后，必须特别小心地删除单引号。应将程序彻底地运行一次以上，以确保在调试过程中跳过的程序段被正确地修复。

§ 8.1.4.3 在指定的检查点打印中间结果

程序员经常发现，程序可运行所有的输入数据，但计算结果不正确。在程序测试过程中，如果在决定进行哪种计算和不进行哪种计算的逻辑流程关键点检查指定变量的内容，将会有所帮助。

在程序中特定的检查点检查某变量的内容有助于逻辑错误的查找。这样，通过对数据进行必要的人工操作，并将结果与各检查点显示的计算机产生的结果相比较，可容易地发现错误。发现偏差时，在前一个检查点之后必定有逻辑错误。

LPRINT 或 PRINT 语句用来在打印机上或屏幕上打印或者显示变量的内容。通常，随显示的数据有简要的说明信息，以帮助用户辨别打印的是哪一个变量。

例 假设要计算数值的总和，检查程序的逻辑错误时，发现总和不正确。下述的PRINT语句将打印各输入值和累加后的总和。

```
LET FinalTotal = FinalTotal + Amt
PRINT "Amt = "; Amt                    'These PRINT instructions are
PRINT "FinalTotal = "; FinalTotal      'for checking intermediate results
```

在调试过程中打印中间结果有助于检查每次的计算结果和查出潜在的错误。

用于这种场合的 PRINT 语句应放在程序中测试指定算术或逻辑指令输出的关键位置。

在上面的例子中，在每计算一次之后，都应通过显示计算机产生的中间结果的办法检查每次的加法。为确保程序的正确运行，人工单步执行程序，并将结果与计算机产生的显示结果相比较。

总之，为调试程序，PRINT 或 LPRINT 语句可用来显示或打印程序中关键检查点的中间结果。在程序充分调试完毕之后，这些语句应删除。

自我测验(一)

1. 编译过程将检查在遵守本语言规则方面所犯的错误，这种错误叫________
2. （T或F）运行时错误在编译过程中检查.
3. （T或F）编译器可检查所有的逻辑错误.
4. （T或F）无效数据跟有效数据一样，都应用来测试程序.
5. 在Turbo Basic程序中插________和________语句可列出所执行程序中部分语句的序号.
6. （T或F）若使用行号，Turbo Basic 则按程序编码顺序执行语句，而不按行号.
7. 给定输入数据：55，87，72和0，下述程序段的跟踪信息是什么?

```
 10  TRON
 20  INPUT ExamGrade
 30  DO  WHILE ExamGrade > 0
 40      IF  ExamGrade > 70 THEN
 50          PRINT "Satisfactory"
 60      ELSE
 70          PRINT "Below C"
 80      END IF
 90      PRINT "Enter exam grade (negative if done)"
100      INPUT ExamGrade
110  LOOP
```

8. 用Trace（跟踪）的方法查出下述程序的错误:

```
10  PRINT
20  BeginningMiles = 75200
30  EndingMiles = 73950
40  FOR I = BeginningMiles TO EndingMiles STEP 200
50      FillUps = FillUps + 1
60  NEXT I
70  PRINT "Number of Fillups "; FillUps
99  END
```

9. 为显示程度段中所设定变量的值，可插入________语句.
10. （T或F）为中止死循环，用Escape键.

§8.2. 通过确认输入避免逻辑错误

§8.2.1 为什么必须输入确认?

定期运行程序时，往往要对由键盘上键入的输入数据或对事先已建好的输入进行操作. 在这两种情况下，数据输入员在开始时输入数据. 因为数据输入员是人，因而录入错误的可能性很大. 设计良好的程序在输入数据按有效数据处理之前包括辨识这种错误的步骤. 也就是说，程序员应在程序中引入错误控制过程（error control procedure），以确认

输入数据，这种过程用来使错误的输入按有效数据处理的风险降至最小。

所有的“实际”程序均应包括如下功能的程序:

(1) 辨别各种可能发生的输入错误;

(2) 列出各个具体的错误，并根据错误情况采取相应的措施;

§8.2.2 无效输入引起的一些后果

§8.2.2.1 输出不准

如果数据录入员为一个工资单打印程序中的 Salary（工资）变量输入 43265，而不是 41265，那么结果将不准。要让程序本身发现这种错误将极其困难。在产生输出时，必须依靠财务部门的职工对这种错误进行复查.

因此，程序不能完全清除导致不准确输出的错误，不过，通过确认输入的格式是否正确和数值是否合理的方法,可以避免将某些类型的错误输入当作有效的数据处理。于是，对于工资变量，检查其是否为数字，也可以检查它的数值是否在正常的范围之内。例如，某公司职员的合理工资范围为 15000–25000，程序应确认每个工资均在此范围之内。这样，键入 257311 或 3287 的工资额就可轻易地辨认为一个错误.

数据确认过程可使错误输入的变量被处理的风险降到最小程度。查出这些错误可降低产生不准确输出的可能性，并改由程序的可靠性.

§8.2.2.2 输入错误导致的逻辑错误

数据验证的一个重要内容是在产生错误输出之前查出输入错误。然而，输入错误常引起运行时错误；这就是说，程序将完全不能运行给定的输入（如，读入为 0 的除数).

假设一个称为 NoofCopies 的输入变量用来确定邮件标签编写循环的运行次数。为此，编程如下:

```
CopiesMade = 0
INPUT NoOfCopies
DO WHILE CopiesMade <> NoOfCopies
    LPRINT ...
    LPRINT ...
    .
    .
    .
    CopiesMade = CopiesMade + 1
LOOP
```

该程序只有在输入变量 NOOfCopies 包含有效数据的情况下，才能正常工作。即，它必须为零或正整数。否则，若 NoOfCopies 为负数，那么它永远也不会等于 CopiesMade。在这种情况下，循环将一遍又一遍地执行，程序进入死循环。这种程序只

能通过按 Ctrl–Break 组合键（若 keyboard break ON 选项已选好的话），或重新启动计算机的方法来中止其运行。因此，这类输入错误导致逻辑错误或运行时错误。

上述例题就是输入无效时可能发生这种逻辑错误的一个例子。

设计良好的程序应验证输入数据，以最大程度地减小这种逻辑错误出现的可能。下述程序在处理之前先验证变量 NoOfCopies：

```
IF  NoOfCopies < 0 THEN
    GOSUB PrintAnError
ELSE
    DO  WHILE CopiesMade < NoOfCopies
        :
    LOOP
END IF
```

上述程序在循环被执行之前先确认 NoOfCopies 非负，因此减少了因错误输入而引起逻辑错误的可能。

Murphy 准则是一条专业程序员非常熟悉的原则：有可能出错的事件，最终将出错。例如,已测试、调试过程序在定期运行时开始出现错误，也不是不可能的。如果程序员未预先处理每种可能的输入错误，那么最终将出现这种情况。一段时间后，若有人输入格式不正确的输入数据，逻辑错误或运行时错误就将出现。

§8.2.3　数据验证方法

本节重点介绍可用来辨认输入错误的程序设计方法。

§8.2.3.1　测试变量以保证格式正确

a. 数字变量

在实际处理输入数据之前，程序应首先确认所有的输入变量均具有正确的数据格式。如果输入到数字变量中的数据出现除数字之外的任何符号（字母、特殊符号、空字符串等），计算机则将它存作 0。

考虑如下情况:

Edit	Run
INPUT QtyOnHand : :	? a

如果用户偶然为 QtyOnHand 变量输入字母 Q 或任何非数字值，那么，不会出现错误信息。计算机简单地将它转化为 0。为减少这种错误，应测试输入数值，确信要处理的数字变量（在本例中为 QtyOnHand）具有一个大于 0 的数值。QtyOnHard、HourlyRata、HoursWorked 均为通常应包含正数的变量例子；若不然，便打印一条错误信息或者要求数据录入员确认其内容。

b. 字符串变量。

假设程序中包含语句：INPUT String $，并且操作员在该字符串变量中实际输入数据，只按 Enter 键。计算机将在 String $ 中存放一个空字符串；这就意味着，没有传递数据，并且字符串被看作空的。虽然空字符串按空格打印，但是，如果进行比较，便可看到它比空格" "小。这样也会导致程序设计错误。在输入字符串变量，如输入 EmpName $ 或 Address $ 之后，应测试该字符串变量，以确保它包含一个"大于"空格的值。

§8.2.3.2 可靠性测试

如前所述，如果输入数据不正确，通常不可能明确地确定。这就意味着虽然程序设计过程可最大限度地减少未检测错误的可能。但是不能指望程序检查所有的错误。许多种包括对数据可靠性的测试。因此，例如，虽然我们不能保证工资变量在输入 3500 的合法性，但肯定可将数值为 998325 的工资变量标作可能的错误。可靠性测试（Test for reasonableness）包括如下方面：

a. 值域测试

确认数据的一种方法是确保变量通过值域测试（Range test）；即，特定变量中包含的数值应在事先规定范围内。如果账号从 00001 编至 85432，还从 87001 编到 89005，那么可以引入一个值域测试，作用可靠怀检查的一部分：

```
INPUT AccountNoIn
IF  (AccountNoIn > 00000 AND AccountNoIn < 85433)_
    OR (AccountNoIn > 87000 AND AccountNoIn < 89006) THEN
    GOSUB ProcessTheData
ELSE
    GOSUB PrintAnError
END IF
```

b. 边界测试

当变量未超出给定的边界时，可进行边界测试（Limit Test）。例如，经理设立了一项制度：手头拥有的任何部件数量不得超过 980 件。当 QtyOnHand 作为输入读取数值时，程序应确保这个界限不得超过。

```
INPUT QtyOnHand
IF  QtyOnHand > 980 THEN
    GOSUB PrintAnError
ELSE
    GOSUB ProcessQtyOnHand
END IF
```

§8.2.4 变量的类型及其影响

§8.2.4.1 概述

前面介绍的在多数例子中已使用过两种类型的变量:

以一个字母或数字作结尾的单精度数值.

或:

以一个美元符（$）作结尾的字符串变量.

其实，可以定义四种基本的数字数据类型:

变量类型	变量名的最后一个字符	例
整型	%	salay%，Day%
长整型	&	Tousalaries&
单精度型	!（或字母，数字）	HourlyRate（或 HourlyRate)
双精度型	#	TotalProfit#

注意: 迄今为止，我们处理过的数字变量要么为单精度数值，要么为整数. 表 8.1 给出有关字符串变量和上述四种数字数据的有关信息(见下页).

§8.2.4.2 小数输入到整型变量时的处理

若指定一个数字变量为整型，则应存放整数值. 考虑如下情况:

Edit

```
INPUT I%
PRINT "The Number is "; I%
```

Run

```
? 3.7
The Number is 4
```

Edit

```
INPUT I%
PRINT "The Number is "; I%
```

Run

```
? 3.3
The Number is 3
```

表 8.1 数据类型

数据类型	有效字符	数值范围	特　性
字符串型 (末尾为 $)	允许使用的任何字符	0～32767 个字符	注意：IBM PC 版本的 BASIC，即 BASICA 允许字符串变量最多包含 255 个字符.
整型 (末尾为%)	全体数字，+、−号（不允许有小数点）	−32768～31767	整型常数必须在开始处使用%（如% Rata＝5）. 常量在程序中不发生变化.
长整型 (末尾为&)	同　上	$-2^{31}\sim2^{31}-1$	长整型的处理速度比普通整型的慢
单精度型 (末尾为 !或字母、数字)	数字、小数点、+、−号	单精度数值的有效数字为六位；六位之后的四舍五入；这可能使数值与实际值相差甚远.	迄今为止，本书已重点介绍了单精度数.
双精度型 (末尾为#)	同　上	双精度具有十六位的有效数字.	双精度数占用较大的内存空间. 双精度数用来存放精度要求在六位有效数字之上的数据.

在此，假定为 I%输入 3.3. 因为 I%是整数变量，所以计算机将输入的值舍入到最近的整数，即 3. 在下述情况下，3.7 将舍入为：4.

总之，计算机将输入的数值舍入到最近的整数. 不过，如果输入 3.5，那么它将舍入到 4；另外，4.5 也将舍入为 4. 这是因为，n.5（n 为任意整数）舍入到最近的偶数.

小　结	
输　入	结　果
为 I%输入 n.y（y＜5）	n 存入 I%
为 I%输入 n.y（y＞5）	n+1 存入 I%
为 I%输入 n.5	若 n 为偶数，n 存入 I%；若 n 为奇数，n+1 存入 I%

§ 8.2.4.3 为数字变量确定数据类型

对于只包含整数的数字变量，应将它明确地定义为整型（变量名的末尾为%），或长整型（变量名的末尾为&）。具体用哪种类型，取决于变量要存放数值的大小。如果整数变量的数值可能大于 32767，或小于−32768，那么将它定义为长整型，用作为其变量名的最后一个字符（如，Amt）；此外，通常将计算器定义为整型.FOR...NEXT 循环也应使用整型变量:

```
FOR I% = 1 TO 3
      :
NEXT I%
```

最后，为所有精度要求在六位有效数字之上的数字变量使用双精度.

注意，可以用 PRINT USING 指令不打印不想要的数字.

§ 8.2.4.4 浮点数

包含小数的数字变量叫做"浮点数"（floating point number）. 在 Turbo Basic 中，所有的单精度和双精度数值均认为是浮点数. 本节将介绍浮点数的实际存储方法.

浮点数由两部分组成:

a. 实际数字（叫做尾数）
b. 指定小数点位置的10的幂次

1234.56 按浮点数表达，则为 1.23456E3. E3 表示将小数点往右移三位便成变量的实际值. 1234.56 也可表示成下列浮点数:

```
.123456E+4
123456E+2
123456E+1
12345.6E−1 (E−1 表示将小数点往左移一位)
123456E−2
```

作为一个约定，Turbo Basic 将带小数部分的浮点数表示成如下形式:

```
n.unn...E+m
```

或:

```
n.nnn...E−m
```

即，2387 表示成 2.387E+3，62 表示成 6.2+1，0.27 表示成 2.7E−1，等等.

不用 PRINT USING 指令显示单精度或双精度的值时，常用浮点记数法显示．变量也可以按这种形式写进文件之中.

Turbo Basic 采用 IEEE 标准表示浮点数，与某些其它版本的 Basic 不同．因此，读取由不同于 Turbo Basic 的某 Basic 版本创建的文件时，必须进行格式转换（参见第十三章）.

§8.2.4.5 定义变量名：一句应认真对待的话

许多程序设计语言，如 Pascal 和 COBOL，要求在程序的数据块中定义变量．Basic 不要求这种明确定义．明确定义的缺少增加了出错的可能.

考虑如下例子：

```
INPUT Amt1
.
.
.
LET Total = Total + Amount1        'Amount1 should be Amt1
```

假设为变量 Amt1 输入数值，但在后来将它错误地引用成 Amunt1（如上所述）．在 COBOL 和 Pascal 语言中，编译时，一个表示 Amount1 未在数块中适当定义的错误将发生．而在 Basic 中，没有明确的数据块；Amount1 将被简单地认为是一个新的变量，其内容为 0．因此，唯一可确定错误已发生的方法在于运行.

因为 Basic 语言不要求进行数据定义，因而引用变量中的这种错误频繁出现．这样，常常导致难于查找的逻辑错误，建议用户在编译和运行程序之前仔细检查所有变量的一致性．同样，在处理之前对数据进行合法性测试，可查获这种错误，并减少调试时间.

§8.2.4.6 减少错误的方法

一种可用来突出变量名，以便在程序易于检查的方法如下：

程序员经常将一系列不仅描述程序逻辑，而且给出使用的变量名及其意义的注释放在程序的前面．这样就可较容易地查出变量名的任何不一致的使用.

自我测验(二)

1. 程序中为何需要错误控制过程?

2. 当整型变量的值输入成带小数部分的浮点数时，小数部分将（截去，四舍五入）。
3. （T或F）设计良好的程序可完全消除错误输入引起的错误。
4. （T或F）输入错误是计算机产生的错误比任何其它类型错误多的原因
5. （T或F）整型变量以% 作结尾。
6. 包含小数的数叫做________。
7. 什么时候应将变量定义为双精度变量?
8. （T或F）如果键入字母A响应输入数字变量值的提示，那么A被转化为0。
9. （T或F） LongNum&定义一个长整型数。
10. 为保证字符串变量含有数据，如何测试?

§8.2.5 用 SELECT CASE 语句确认数据

回忆一下，程序中有四种逻辑控制结构：顺序、选择、迭代和分情形（case）结构。在 Turbo Basic 中，用 SELECT CASE 语句实现分情形结构（case structure）。SELECT CASE 语句可测试一系列用于数据确认子程序中的条件。

假设一个名字为 YearsInCollege 的输入变量用于确定将进行处理的类型。有：

```
IF  YearsInCollege = 1 THEN
    GOSUB ProcessFreshman
END IF
IF  YearsInCollege = 2 THEN
    GOSUB ProcessSophomore
END IF
IF  YearsInCollege = 3 THEN
    GOSUB ProcessJunior
END IF
IF  YearsInCollege = 4 THEN
    GOSUB ProcessSenior
END IF
```

为确保处理正确，必须加入第五个条件，以便输入变量无效时执行错误处理子程序。即，如果它的值为 1～4 之外的任何值，则将它看作一个错误：

```
IF  YearsInCollege NOT = 1 AND YearsInCollege NOT = 2_
    AND YearsInCollege NOT = 3 AND YearsInCollege NOT = 4 THEN
    GOSUB PrintAnError
END IF
```

SELECT CASE 语句可使得一系列情形更清楚地编程，并写成结构化形式。

例 1

```
INPUT YearsInCollege
SELECT CASE YearsInCollege
    CASE = 1
        GOSUB ProcessFreshman
    CASE = 2
        GOSUB ProcessSophomo
    CASE = 3
        GOSUB ProcessJunior
    CASE = 4
        GOSUB ProcessSenior
    CASE ELSE
        GOSUB PrintAnError
END SELECT
```

当 YearsInCollege 不等于 1，2，3 或 4 时，执行语句： GOSUB PrintAnError。该 Case 结构的伪代码如下：

```
CASE   YearsInCollege
    WHEN  1       执行 Freshman 子程序
    WHEN  2       执行 Sophomore 子程序
    WHEN  3       执行 Junior 子程序
    WHEN  4       执行 Senior 子程序
    WHEN  其它值   执行错误处理子程序
END  CASE
```

SELECT CASE 指令有如下格式：

```
SELECT  CASE   表达式
    CASE  测试表
           语句组
   [CASE  测试表
           语句组]
             ⋮
   [CASE  ELSE
           语句组]
END  SELECT
```

其中，语句组必须与 CASE 行分开，不能同占一行．例如：

```
CASE = 1   GOSUB   ProcessFreshman
```

所有元素均位于同一行将导致一个错误.

"CASE 测试表"的例子有:

```
CASE  = 5              ' 关系测试
CASE  = 18             ' 相等测试
CASE  10  TO  99       ' 范围测试
CASE  10, 20           ' OR (或) 测试 (如, = 10 OR 20)
```

例 2 假设要用 CASE 结构根据学生的平均成绩打印班级的等级:

```
INPUT Average
SELECT CASE Average
    CASE 90 TO 100
        PRINT "A"
    CASE 80 TO 89
        PRINT "B"
    CASE 70 TO 79
        PRINT "C"
    CASE 60 TO 69
        PRINT "D"
    CASE 1 TO 59
        PRINT "F"
    CASE ELSE
        PRINT "ERROR"
END SELECT
```

总之, SELECT CASE 指令可用来测试一系列条件的结果. 它有数不清的应用场合, 并常用于数据的确认. 即, 先用 SELECT CASE 语句测试变量中所有的有效输入, 然后引用 CASE ELSE 子句确定变量中是否有无效的数值. 若有无效数值, 则执行错误处理子程序.

§ 8.2.6 数据确认的其它方法

§ 8.2.6.1 用控制表或跟踪记录人工确认输入数据

虽然计算机错误通常源于错误的输入, 但它们也可能是由于对公司的蓄意捣乱或欺骗而引起.

处理大型文件时, 一种减少任何未发现错误的方法是: 打印一份控制表 (Control listing) 或跟踪记录 (audit trail). 其中包括:

(1) 用于所处理的各输入记录的标识数据或关键变量;

(2) 所遇到的任何错误;

(3) 所处理的输入记录组的累计总和.

通常, 由用户方的人负责检查该控制表, 以确保所进行的处理正确的任务. 图 8.3 是一张描述控制表样例的打印格式图. 因为磁盘不可直接阅读, 所以创造一个输出磁盘文件时, 大多数机构要求有控制表.

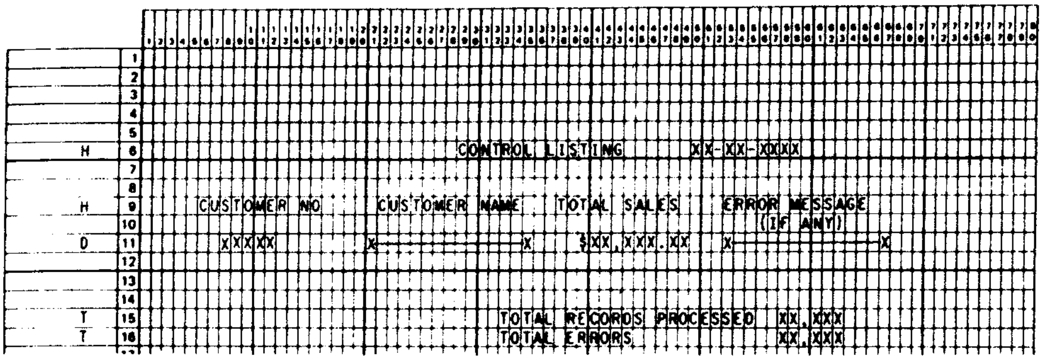


图 8.3 控制表样例的打印格式图

§8.2.6.2 核实或回显的方法确认输入的数据

减少数据录入错误的另一种方法是：用重录或录入检验过程（key verification procedure）的方法核实所有的输入数据．如果数据录入员被要求重新输入已录入的数据，那么程序可检查输入情况，看刚开始输入的数据是否与第二次输入的数据相同．若不相同，则核实数据的操作员必须查出每一个错误，并将它修正．

另一种确认数据的方法是让操作人员输入数据，将它送到计算机，并让计算机"回显(echo)"或再次显示；然后，程序要求操作员核实所传送是否正确．例如：

```
LET Verify$ = "Y"
DO  WHILE Verify$ <> "N" AND <> "n"
    PRINT "Enter Name:"
    INPUT EmpName$
    PRINT "Enter Amount:"
    INPUT Amount
    PRINT "Name is "; EmpName$; " and Amount is "; Amount
    PRINT "Is this correct (Y/N)?"
    INPUT Verify$
LOOP
```

确认过程将查出所有数据输入错误的 90%左右．虽然这个过程可使问题降至最小程度，但仍有 10%左右的输入错误未能查出．联合进行人工检查和程序控制将查出绝大多数，或所有的输入错误．

§8.3 输入错误的处理

本节给出输入错误发生时可能要采取的几个措施．

§8.3.1 中止程序的运行

出现大错误时，最简单的方法是用 STOP 指令中止程序的运行．STOP 指令在数据的

完备性是主要的考虑因素，并且错误必须保持绝对少时编入。通常，应当有一名用户方面的职员负责数据的检查；他（她）必须修正数据错误，并安排重新运行事宜。

例如:

```
IF  Total > 100000 THEN
    PRINT "Total Exceeds 100000--Run Terminated"
    STOP
END IF
```

§8.3.2 打印一个包含关键变量、出错变量内容和错误信息的错误行

在错误行中，错误应总是和相应的错误信息一起清晰地显示；此外，还必须包含标明错误数据的关键变量。例如:

社会保障金号	职工姓名	错误
080—65—2113	BROWN JOSEPH	工资项为空
092—11—7844	LOPEZ MARIA	婚姻状态项无效(=0)

应保存各种类型错误发生次数的统计结果。用户可用该统计项确定是否存在任何大的程序设计或系统问题。也就是说，如果某类型错误出现的次数太多,则可能存在程序错误，或数据录入员重复犯某种特定的错误。

§8.3.3 要么部分处理错误数据，要么整个跳过错误数据

一旦被发现错误，程序就应该采取如下两种方法中的任一种:

(1) 整体跳过出错的记录或变量，开始处理下一条记录或下一个变量。
(2) 处理出错记录或变量的某些部分。

这也是用户需要的功能。例如，有的时候错误输入的数据变量应替换成零；而另一些时候，它又应简单地忽略。

§8.3.4 错误数目超过预定的极限时中止运行

有时，即使出现了错误，仍希望继续处理其余的数据；但这种错误太多时，就必须中

止程序的运行．考虑如下错误处理子程序:

```
ErrorProcedure:
    PRINT
    .
    .
    .
    NoOfErrors = NoOfErrors + 1
    IF  NoOfErrors > 25 THEN
        PRINT "Run Terminated--Too Many Errors"
        STOP
    END IF
RETURN
```

§8.3.5 用开关"标记"错误

假如要对各记录或变量进行多次确认测试，但希望只处理有效数据；即没有任何错误的变量或记录，这样就必须使用一个开关（switch)．开始时，将开关变量 ErrSwitch 初始值设为 0．如果发生错误，则在错误处理子程序中将它置为 1．

待完成所有的确认测试之后，再测试 ErrSwitch．如果为 1，那么它表明至少发生了一个错误，可进行相应的处理．否则，它为 0 时，则表明没有错误发生．在处理各个新记录或变量之前，保证将 ErrSwitch 重新置成 0．

根据错误是否发生，变量 ErrSwitch 将等于 0 或 1．

```
ValidityTests:
    .
    .
    .
    IF  ErrSwitch = 1 THEN
        LET ErrSwitch = 0       'Reinitialize ErrSwitch
    ELSE
        GOSUB ProcessTheRecord
    END IF
RETURN
```

or

```
ValidityTests:
    .
    .
    .
    IF  ErrSwitch = 0 THEN
        GOSUB ProcessTheRecord
    ELSE
        LET ErrSwitch = 0       'Reinitialize ErrSwitch
    END IF
RETURN
```

§8.3.6 处理输入文件或大量的变量时打印控制总数

§8.3.6.1 打印所有错误和所有记录或变量的统计总数

程序应提供所处理记录或变量，以及所发生错误的统计数字．为了确定所处理记录或变量的数目，每处理一次记录或变量均使总数加 1，然后在工作结束时打印总的统计数字．对于大多数生产运行过程，在开始处理之前用户负责统计输入的变量数或记录数．然后，用户将人工列表的总数与计算机统计和实际处理的变量或记录总数进行比较．两者应当相等．如果不等，记录或变量则可能输入不正确．此时，用户必须找出错误，并将它修正．

§8.3.6.2 打印分类总数

如果在运行期间要处理大量的输入记录或变量，所有数据的单一统计总数则可能不足以跟踪出错数据．在这种情况下，应引用分类总数（batch total）其中包含指定组或批中所有变量或记录的统计总数．假设要根据业务号 TransNo 处理业务，那么可能有如下的分类总数:

所处理记录的总数	1130
TransNo 为 001–082 的记录数	48
TransNo 为 083–115 的记录数	53
TransNo 为 116–246 的记录数	387
TransNo 为 247–383 的记录数	226
TransNo 为 384–452 的记录数	417

每一个单独的总数被称为一个分类总数．在输入数据被录入之前，用户确定要处理的各类记录或变量．人工统计的分类总数应与计算机生成的分类总数相匹配．否则，用户应找出特定组中未处理的数据．

程序本身也可用来比较人工统计的分类总数和计算机产生的分类总数．为此，人工统计的分类总数值随输入数据一起输入．将人工统计的分类总数作为文件数据输入的方法有:

(1) 作为一条控制记录：它很可能是包含分类总数信息文件中的第一条记录．

(2) 作为一条带定界符的记录；它包含各组或各类的人工统计总数．

(3) 在键盘上交互输入分类总数．

§8.3.7 发现错误时让用户中止处理

错误出现时中止处理的程序叫做“错误处理程序（error handler）”．下面介绍 ON KEY 错误处理程序．

ON KEY 错误处理程序

可以在程序中引入指令，以便用户按一个功能键时中止所进行处理:
下面的两条指令联合用于这个目的:

```
:
KEY(n) ON     'n is a number from 1 to 10--it refers to F1-F10
:
ON KEY(n) GOSUB label
```

在 KEY(n) ON 语句中，n 可以是 1～10 之间的数值．KEY(n) ON 语句启动检查功能键 F1–F10 中任一键是否被用户所按的功能．即，如果在程序中编写 KEY(6) ON，那么该程序可检查功能键 F6 是否被按．KEY(n) ON 启动检查特定功能键的功能．测试功能键是否被按下的实际指令为: ON KEY(n) GOSUB 指令．

```
ON KEY(6) GOSUB ErrorDetect
```

若 F6 被按，便执行 ErrorDetect 子程序
如果用户按其功能键，ON KEY 错误处理程序便中止所进行的处理．用户既可以在此处中止处理，也可以在程序中的其他模块内继续处理．
考虑如下程序:

```
KEY(1) ON        'Set function key F1 for error trapping
PRINT "Press the F1 key to terminate processing"
:
DO  WHILE Amt < 999
    :
    ON KEY(1) GOSUB TerminateTheRun
    :
LOOP
END
```

如前所述，KEY(n) ON 和 ON KEY(n) GOSUB 语句的 n 可以为 1～10，分别代表功能键 F1～F10．它也可以为这里未介绍的其它值（详见附录）．

8-320 B25

§ 8.4 什么时候应确认数据?

所有定期运行的程序都应引用旨在减少错误的数据确认技术. 从此以后, 我们在考产程序段时都将介绍这种技术. 但各章末尾的实例部分将重点描述本章的主题; 并为简便起见, 不着重考虑数据确认方法. 不过, 各章末尾的复习题将要求读者修改实例, 以包括相应的数据确认子程序.

§ 8.5 用于调试和数据处理的 READ 和 DATA 语句

§ 8.5.1 READ 和 DATA 语句的用法: 概述

迄今为止, 我们已重点介绍了用于数据输入的 INPUT 语句. 固定的数据也可以作为程序的一部分输入,并用 READ 和 DATA 语句读入变量. 由于 READ 和 DATA 语句可在调试阶段用作 INPUT 语句的替代语句, 因此本章将介绍这两个语句. 正如我们在后面章节中将看到的一样, 它们还有许多其它用途.

READ 语句和 DATA 语句按下面例子中的形式配对使用:

```
READ Amt1
READ Amt2
READ Amt3
    .
    .
    .
DATA 20.3, 15, 4.6
```

第一个 READ 语句从第一个 DATA 语句中读取第一个未读过的常数, 并将它赋给指定的变量. 各后续 READ 语句按同样方法处理. 因此, 在上例中头三个 READ 语句分别将 20.3、 15、 4.6 送到变量 Amtl1, Amtl2 和 Amtl3 中. 这样, 数据可直接在程序中确定, 并赋给任意位置的变量.

下述程序读取三组两个变量, 并将它们相加, 然后打印各个总和:

```
FOR I% = 1 TO 3
    READ Amt1, Amt2
    LET Total = Amt1 + Amt2
    PRINT USING "###.#"; Total
NEXT I%
DATA 46.3, 92.1, 88, 72, 4, 3.6
```

其打印结果为:

138.4 ← 46.3 和 92.1 之和 (Amtl1 = 46.3, Amlt2 = 92.1)

```
160.0   ←  88 和 72 之和 (Amt1 = 88, Amtl2 = 72)
  7.6   ←  4 和 3.6 之和 (Amtl1 = 4, Amtl2 = 3.6)
```

即，第一次执行 READ 语句时，46.3 被送入 Amtl1，92.1 被送入 Amtl2；第二次执行 READ 语句时，DATA 语句中的下两个数值 88 和 72 被分别送入 Amtl 和 Amtl2；最后，第三次执行 READ 语句时，DATA 语句中的下两个数值：4 和 3.6 被分别送入 Amtl1 和 Amtl2。

一条 READ 语句可以与多个 DATA 语句配合使用。下述段序段产生上面例子同样的结果。

```
FOR I% = 1 TO 3
    READ Amt1, Amt2
    LET Total = Amt1 + Amt2
    PRINT USING "###.#"; Total
NEXT I%
DATA 46.3, 92.1                 'We may use 3 DATA statements,
DATA 88, 72                     'one for each pair of amounts
DATA 4, 3.6                     'read
```

第一次执行 READ 语句时，从第一条 DATA 语句中读取数据。如果要读入条目的个数比第一条 DATA 语句中列举的数据个数多，那么计算机接着从后面的 DATA 语句中赋给数值。

```
FOR I% = 1 TO 10
    READ Amt1, Amt2
    LET Total = Amt1 + Amt2
    PRINT USING "###.#"; Total
NEXT I%
DATA 46.3, 92.1, 88, 72, 4, 3.6, 106.8, 202.3, 5, 6
DATA 19.6, 12, 14.3, 11, 9, 2.6, 88, 72, 88, 100
```

刚开始执行 READ 语句时，计算机查找第一个 DATA 语句，并将刚开始的数值送入 READ 语句列出的第一个变量。

字符串变量和数字变量一样，均可用于 READ 语句中。例如：

```
READ EmpName$, Salary
:
:
DATA Mickey Mouse, 15500
```

DATA 语句可放在程序中 READ 语句之后的任何地方。作为一个约定,将 DATA 语

句放于所引用模块的最后.

避免常见的错误

使用 READ 和 DATA 语句时，最常发生的错误是无意中将字符串数值送入数字变量. 若发生这种错误，将出现第 2 号运行时错误.

DATA 语句中数值的个数必须足以满足 READ 语句的需要. 下述程序将产生第 4 号实时错误: "Error 4:　Out of Data (数据不足)".

```
FOR I% = 1 TO 3

    READ Amt1, Amt2
    LET Total = Amt1 + Amt2
    PRINT Total
NEXT I%
DATA 55, 22, 11, 19, 21        'There must be six values here
                               'or another DATA statement
```

RESTORE 恢复语句重新设置 DATA 语句中的数值，以便 RESTORE 语句之后的第一个 READ 语句重新从第一个 DATA 语句中读取数据:

```
FOR I% = 1 TO 5
    READ Amt1, Amt2
    LET Total = Amt1 + Amt2
    PRINT Total
NEXT I%
RESTORE
FOR I% = 1 TO 5
    READ Amt1, Amt2
    LET Total = Amt1 * Amt2
    PRINT Total
NEXT I%
DATA 5, 8, 7, 2, 1             'These numbers
DATA 10, 16, 12, 4, 3          'are processed twice
```

该程序为 Amtl1 和 Amtl2 处理五组数据. 首先，它累加各组数值并打印总和; 然后即在 RESTORE 语句之后，将各组数据相乘，并打印结果.

自我测验(三)

1. 什么时候应使用SELECT CASE语句?
2. 列举关键字段或关键变量、错误和分组总数的打印输出叫________或________.
3. 为一个表示婚姻状况的编码写一个SELECT CASE结构; 其中:

1　表示未婚.
2　表示已婚.
3　表示分居.
4　表示离婚.
5　表示寡居或鳏居.
其它数值均无效.

4. 重新键入输入数据以确保第一次输入正确的过程叫________过程.
5. 编程让计算机读取售货员姓名和数量，然后"回显"输入数据.
6. (T或F) 发现输入错误时，程序员总是要中止程序的运行.
7. 可用于对各记录进行多次确认测试，以保证发生错误时可跳过某过程或程序段的时候.
8. 编程计算事务文件中错误记录的数目. 对各记录进行如下测试:

社会救济金额必须为正整数;
雇员姓名必须给出
所得税必须为＄4.00～＄25.00之间的数.

9. 修改第8题，还计算所处理的所有有效记录数目.
10. 语句可使用户通过按功能键的方法中止所进行的处理.

§8.5.2　什么时候使用READ和DATA语句?

READ和DATA语句最好用于:

1. 调试期间替代INPUT语句.
2. 输入基本固定偶而变化的数据.

§8.5.2.1　在调试期间替代INPUT语句

调试程序时，用户可能更喜欢将数值存入DATA语句，然后用READ语句进行处理. 这样，在调试时不必响应输入提示. 此外,它还可以按用户要求设定数据精度，以便人工确定输出的形式. 此后，用户应仔细检查计算机产生的输出以确保它的正确性.

在调试过程中若用了READ和DATA语句，则调试阶段结束时通常要删除这些语句，并在最终的检查中使用所要求的INPUT语句. 最后的运行测试确保实际输入数据的正确性.

§8.5.2.2 输入基本固定仅偶而变化的数据

有些用于程序之中的数据完全固定，并可设为常数；其中包括月号（01–12），月名(Jan–Dec）和公司名称等．然而，某些元素目前是常数，但常随时间变化；其中包括销售税率、社会救济金税率、指定物品的单价等．在第十章中，我们可输入表格中的这种条款；表中输入数据在每次运行时装入．另一种方法是，用 READ 和 DATA 语句将这种条款送入变量．诸如税率之类的变量最好按如下形式用 READ 和 DATA 语句进行初始化:

```
1.  READ  SocSecTaxRate
          :
     DATA  .0751
2.  READ  NYCSalesTaxRate
          :
     DATA  .0825
```

在处理过程中，我们将根据它们的名字引用这些数率．这样，如果税率发生变化，则只需简便地修改 DATA 语句，而不必改动程序中的其它元素．

假设不用这个方法，而在程序中自始至终直接使用 New York（纽约）市的税率 0.0825，需要用新税率替换所有的 0.0825．因为我们可能偶而忘记作出某种修改和或改变了纽约市税率之外的数量．因此这实质上是一种冒险．为了减少用这种方法作出错误修改的风险，最好使用象 NYCSalesTax 之类的变量名，并应用 READ 和 DATA 语句，给它赋值．

§8.6 程序设计特性的回顾

本节复习使程序易于调试和易于阅读、编写及修改的设计思想

一、总以规划或设计作为程序设计的开始．

1. 用伪代码描述程序逻辑
2. 与用户和系统分析员一起讨论伪代码，确保理解所有的设计要求．

二、在程序中采用结构化设计思想和逻辑控制结构

1. 完全避免GOTO语句或分支（本书不介绍GOTO语句）
2. 用四个逻辑控制结构执行所有的指令:
   a. 顺序结构–按顺序执行指令．
   b. 选择结构–根据特定的条件上否满足，用IF–THEN–ELSE–END IF条件块选取一条执行路径．

c. 迭代结构-用DO WHILE....LOOP 或其它循环方法重复执行一组指令.

d. 多情形结构-根据一组情况或条件中的一个是否满足，用SELECT CASE语句执行相应的指令

3. 使程序模块化

a. 执行特定功能的每个设计良好的程序段或语句系列应编作单独模块中的一个单元.

b. 用GOSUB执行各模块（第十一、十二两章讨论其它使程序模块化的方法.）

差的程序形式

```
IF   Amt1 = Amt2 THEN
     PRINT ...
     LET ...
     INPUT ...
ELSE
     PRINT
     LET
     .
     .
     .
END IF
```

优选程序形式

```
IF   Amt1 = Amt2 THEN
     GOSUB EqualRtn
ELSE
     GOSUB UnequalRtn
END IF
```

4. 用自顶而下的方法编程

a. 先编主模块，先编辅助模块. 即，模块应按详细程序的倒序编码.

b. 这种方法使用中首先将重点放在程序结构，然后再填充细节.

5. 确信程序被良好地归档，且易于阅读.

a. 引入注释和／或说明:

```
(1)  REM This is a comment
       .
       .
       .
     INPUT
```

或:

```
(2)  comments begin with an open guote
```

或:

(3)

```
'**********************************
'* This is a long comment         *
'* which is highlighted by being *
'* boxed with asterisks (*'s)     *
'**********************************
```

b. 为易于阅读，缩进子句.

```
IF ...
    PRINT ...
    LET ...
    .
    .
    .
ELSE
    .
    .
    .
END IF
```

c. 一行编写一条语句.

6. 用意义明确的名字标识变量和子程序

§8.7 本章辅导

§8.7.1 本章小结

一、程序错误的种类

1. 语法错误；这类错误必须在程序可完全执行这前修正.
2. 逻辑错误；用Trance选项查找这类错误.

二、确认数据以减少错误

1. 错误处理过程将减少错误，但不能将它完全消除.
2. 错误处理过程的种类.
   a. 值域测试-确保数据位于预定的范围之中.
   b. 值域测试-确保数据不超出预定的界限.
   c. 出错数据测试-确保所有关键包含非零，非空或非空数据.
   d. 用SELECT CASE 测试所有的有效输入,并为无效输入执行错误处理子程序.

三、检测输入错误的方法

1. 打印要用户检查的控制表
   a. 控制表包含识别所处理输入的数据.
   b. 控制表还提供记录的总数和所遇到的任何错误.

2. 确认输入数据

   a. 键入的数据必须通过重新录入或确认过程复查.

   b. 如果重录产生与最初输入数据不同的输入，则必须查出各个偏差的原因，并采取修正措施.

四、处理输入错误的方法

1. 出现大错误时，中止运行.
2. 打印描述所有错误的错误信息.
3. 用空格或零填入出错的变量
4. 统计错误发生次数，并当错误数目过大时中止运行.
5. 允许用户用诸如ON KEY 语句之类的错误程序人为地中止运行. 即，用户按特定的功能键时，程序中止运行.

§8.7.2 本章自测

1. (T或F) 如果程序编写合适，那么所有的输入错误都可查出.
2. ________是一个描述所处理的输入数据和所发现错误的打印报告.
3. 重新录入输入数据以确保第一次输入正确的过程，叫________.
4. 以%终结的变量，始终存放________值.
5. 55按浮点格式表示成________.
6. 如果数字变量包含的数值大于六位有效数字，那么它应说明为________.
7. 指定组中所有记录的数目被称为________总数.
8. 考虑如下程序

```
KEY(4) ON
:
ON KEY(4) GOSUB XYZ
```

   xyz 被执行的前提是________.

9. 下面程序有什么错误?

```
Verify$ = "Y"
DO  WHILE Verify$ <> "N" OR Verify$ <> "n"
    INPUT ...
    :
    PRINT
    INPUT Verify$
LOOP
```

10. 输入一个财产的结构类型，以确定它是否有效．有效的结构类型如下：

1	居住
2	商用
3	闲置

所有其它代码均无效．除结构类型之外，财产的税率也应输入．验证输入之后，用 SELECT CASE 语句编程确认输入，并进行相应的处理（执行相应的子程序）．

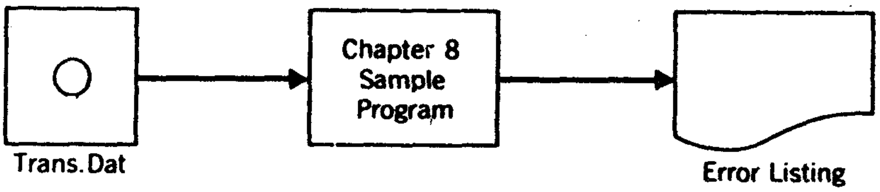
§8.7.3 实例

考虑如下程序结构图

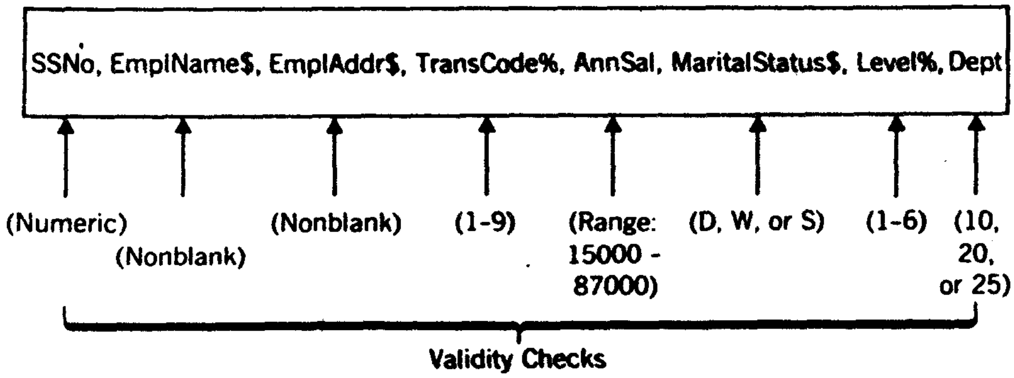
输入数据用 READ 和 DATA 语句输入，各组数值均包括：SSNO，EmplName $，EmplAddr $，TramsCode%，AnnSal，MaritalStatus $，Level%，Dept．

问题描述如下：

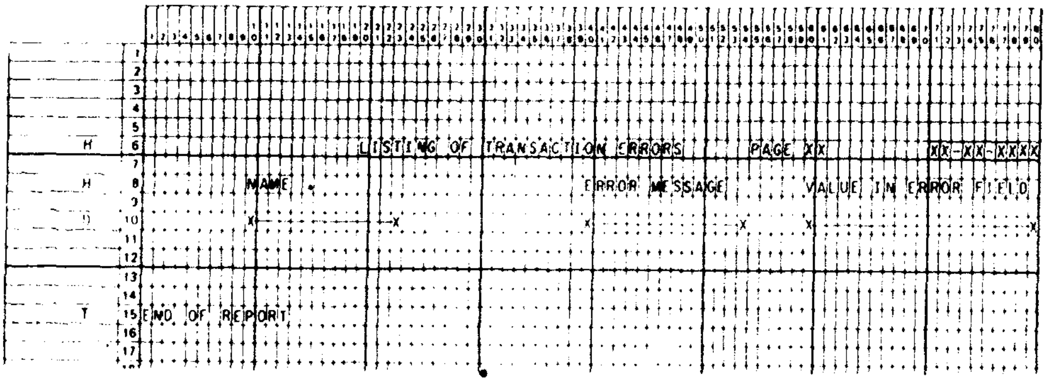
系统流程图



Trans.Dat 的格式（用 READ 和 DTAT 语句读入）



错误清单打印格式图



伪代码和层次图见图 8.4；程序见图 8.5

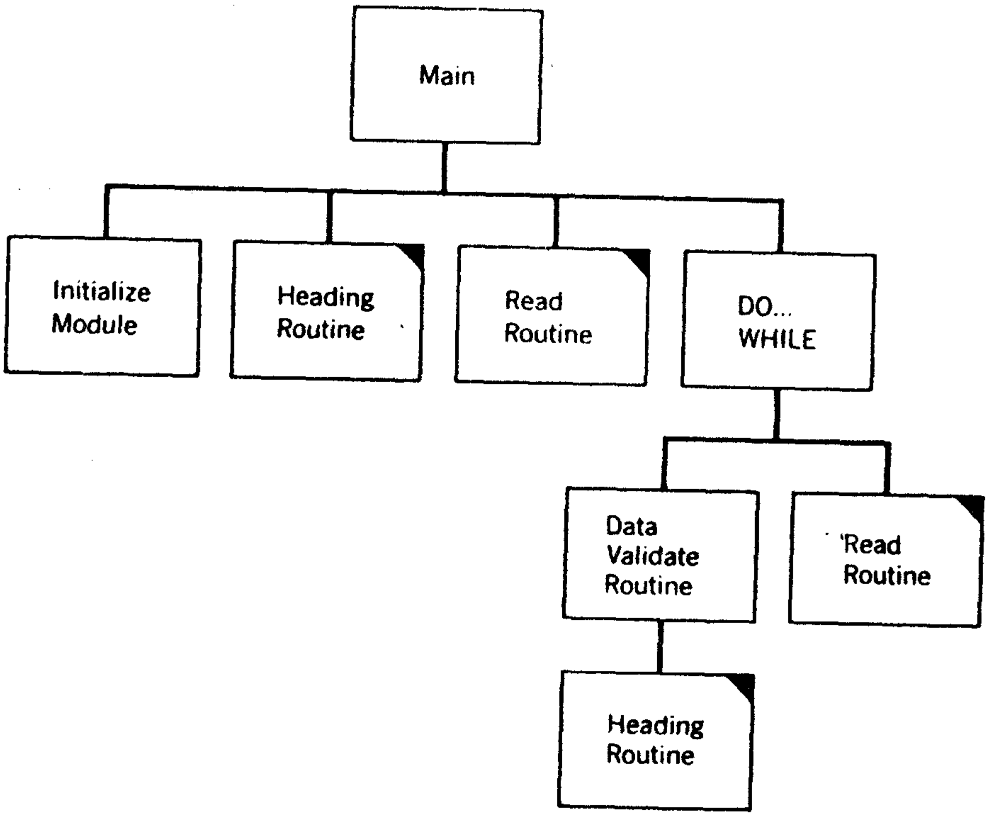


图 8.4(a) 层次图

```
START
    Set printer width to 100
    PERFORM
        Set Date
        Set PageNo to 0
    ENDPERFORM
    PERFORM
        Read data
    ENDPERFORM
    DO  WHILE there is still data
        PERFORM
            Check for validity error
            Set flag to 1 if error occurs
            Print a message if error occurs
            Increase LineCounter
            IF  LineCounter > 25 THEN
                PERFORM
                    Print Headings
                    Reset LineCounter to 0
                ENDPERFORM
            ENDIF
        ENDPERFORM
        PERFORM
            Read data
        ENDPERFORM
    ENDDO
    PRINT "End of Report"
STOP
```

图 8.4(b) 伪代码

```
'Example of a Data Validation Program
WIDTH "LPT1:",100    'assuming printer can print 100 chars per line
GOSUB InitializeModule
GOSUB HeadingRoutine
GOSUB ReadRoutine
DO WHILE SSNo <> 0
        GOSUB DataValidateRoutine
        GOSUB ReadRoutine
LOOP
LPRINT "End of Report"
DATA 080243[illegible],Paul Newman,"11 Main St., NYC",1,18000,[illegible],10
DATA 070263442,Linda Gray,"[illegible] Spring St., NYC",0,[illegible],D,5,20
DATA 11[illegible]47892,Robert Redford,"50 3rd Ave., NYC",[illegible],12000,S,6,25
DATA 048239261,Joan Collins,"20 Sutter Pl., NYC",8,86000,0,5,10
DATA 070235826,Larry Hagman,"40 21 3rd St., NYC",9,45000,R,7,10
DATA 072487[illegible]31,John Smith,"41 3rd Ave., NYC",9,45000,5,6,14
DATA 0,ABC,ABC,9,9,A,9,9
END
 ******************************************************************
  this module initializes date and page number
 ******************************************************************
InitializeModule:
        DateOfRun$ = Date$
        PageNo = 0
RETURN
 ******************************************************************
  this module reads data using READ and DATA statements
  and resets ErrorFlag
 ******************************************************************
ReadRoutine:
        ErrorFlag = 0
        Read SSNo,EmplName$,EmplAddr$,TransCode%,Annual,
                          MaritalStat$,Level%,Dept
RETURN
```

```
***********************************************************
                 This module prints headings
***********************************************************
HeadingRoutine:
        PageNo = PageNo + 1
        LPRINT TAB(20);"LISTING OF TRANSACTION ERRORS";
        LPRINT TAB(55);"PAGE ";
        LPRINT TAB(60);
        LPRINT USING "##";PageNo;
        LPRINT TAB(72); DateOfRun$
        LPRINT
        LPRINT TAB(10);"NAME";
        LPRINT TAB(40);"ERROR MESSAGE";
        LPRINT TAB(60);"VALUE IN ERROR FIELD"
        LPRINT
        LineCounter = 0
RETURN
***********************************************************
  This module checks data for validity and prints any
  errors.  It also sets ErrorFlag to 1 if an error occurs
***********************************************************
DataValidateRoutine:
        IF SSNo < 0 OR SSNo > 799999995 THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"SS NO IS INVALID";
                LPRINT TAB(75);SSNo
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF EmplName$ <= "   " THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"NAME IS INVALID";


                LPRINT TAB (76); EmplName$
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF EmplAddr$ <= "   " THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"ADDRESS IS INVALID";
                LPRINT TAB (76);EmplAddr$
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF TransCode% < 1 OR TransCode% > 5 THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"TRANS CODE IS INVALID";
                LPRINT TAB(75);TransCode%
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF AnnSal < 15000 OR AnnSal > 87000 THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"SALARY IS INVALID";
                LPRINT TAB(75);AnnSal
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF MaritalStat$ = "M" OR MaritalStat$ = "D"
                            OR MaritalStat$ = "S" THEN
                EXIT IF
        ELSE
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"MARITAL STATUS IS INVALID";
                LPRINT TAB(76);MaritalStat$
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
```

```
        IF Level% < 1 OR Level% > 6 THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"LEVEL IS INVALID";
                LPRINT TAB(75);Level%
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF Dept = 10 OR Dept = 20 OR Dept = 25 THEN
                EXIT IF
        ELSE
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"DEPT IS INVALID";
                LPRINT TAB(75);Dept
                LPRINT
                ErrorFlag = 1
                LineCounter = LineCounter + 1
        END IF
        IF ErrorFlag = 0 THEN
                LPRINT TAB(10);EmplName$;
                LPRINT TAB(40);"A-OK";
                LPRINT TAB(75); " "
                LPRINT
                LineCounter = LineCounter + 1
        END IF
        IF LineCounter > 25 THEN
                GOSUB HeadingRoutine
        END IF
RETURN
```

图 8.5(a)　解答

```
             LISTING OF TRANSACTION ERRORS       PAGE  1         02-15-1990
NAME                             ERROR MESSAGE          VALUE IN ERROR FIELD

Paul Newman                      A-OK

Linda Gray                       TRANS CODE IS INVALID           [illegible]

Robert Redford                   SALARY IS INVALID               12000

Joan Collins                     MARITAL STATUS IS INVALID       0

Larry Hagman                     MARITAL STATUS IS INVALID       R

Larry Hagman                     LEVEL IS INVALID                7

John Smith                       DEPT IS INVALID                 14

End of Report
```

图 8.5(b)　打印结果

§8.7.4　复习题

一、是非题

1. 统计交易号为1～100，101～200等的记录数时，可以用分类总数.
2. 控制表可有可无，而且实际上极少由程序产生.
3. 考虑如下语句

Rata% = 5

Rata%被看作整数常数.

4. 在Turbo Basic中，如果编有行号，则指令按行号顺序执行.
5. 运行时错误将导致计算机中止运行.
6. 数据确认极少用来减少错误，因为它只清除为数很小的数据输入错误.
7. 除零将导致语法错误或逻辑错误的出现.
8. 程序中若使用Trace选项，则应使用行号.
9. 字符串比较时,空字符串比空格小("<").
10. 要求键入数字输入时，如果用户键入一字母字符，那么数字变量将等于0 .

二、综合题

1. Trace 选项的作用是什么?
2. 输入数据为何要确认?
3. 列出至少三种数据确认方法，并作出解释.
4. 说明单精度数和双精度数之间的区别.
5. 为什么计数器要定作整型变量?

用浮点记数法表示下列数值：

6. 107.6
7. 0.33
8. 44728
9. 4478.33
10. −26.3

§8.7.5 调试练习

对下述程序进行调试:

```
CLS
PRINT "Enter Transaction Number, Amount, and Type"
INPUT TransNo, AmountTrans, TransType$
HoldTrans = TransNo
GOSUB PrintHeadingsRoutine
DO  WHILE TransNo <> 0
    GOSUB ProcessTransactionRoutine
    PRINT "Enter Transaction Number, Amount, and Type"
    INPUT TransNo, AmountTrans, TransType$
LOOP
PRINT "End of Report"
END
```

```
PrintHeadingsRoutine:
    LPRINT TAB(30); " Transaction Exception Report"
    LPRINT
    LPRINT TAB(20); "Transaction Number";
    LPRINT TAB(40); "Amount";
    LPRINT TAB(50); "Type";

    LPRINT TAB(60); "Message"
    LPRINT
ProcessTransactionRoutine:
    IF  TransNo < HoldTrans THEN
        GOSUB SequenceErrorRoutine
    END IF
RETURN
SequenceErrorRoutine:
    LPRINT TAB(20); TransNo;
    LPRINT TAB(40); Amount;
    LPRINT TAB(50); TransType$;
    LPRINT TAB(60); "Sequence Error"
    LPRINT
RETURN
```

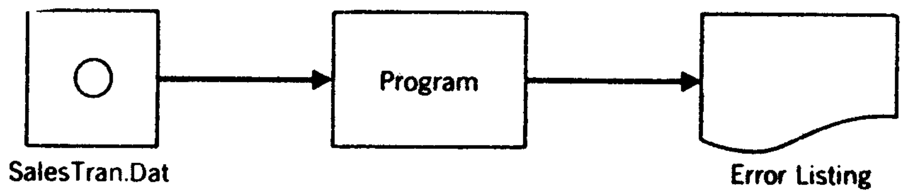
上面程序不能正确运行.

1. 为什么只处理第一条记录?
2. 修正上述程序以便打印报告列出所有未按升序排列的交易.
3. 后省去RETURN语句，将出现什么情况?
4. 只有交易类型"D"和"W"才有效.加入程序检测并打印带错误交易类型的记录.
5. 交易量必须为正数. 加入子程序检测，并打印其交易量为0、或为负的记录.

§8.7.6 编程题

1. 编程检查销售记录是否按售货员号的升序排列. 问题描述见图8.6.
   注意: 为任何非按顺序排列的记录，打印售货员号及其姓名.

系统流程图



SalesTran.Dat 的记录格式

SalesPersonNo, SalesPersonName, OtherData

错误清单打印格式图

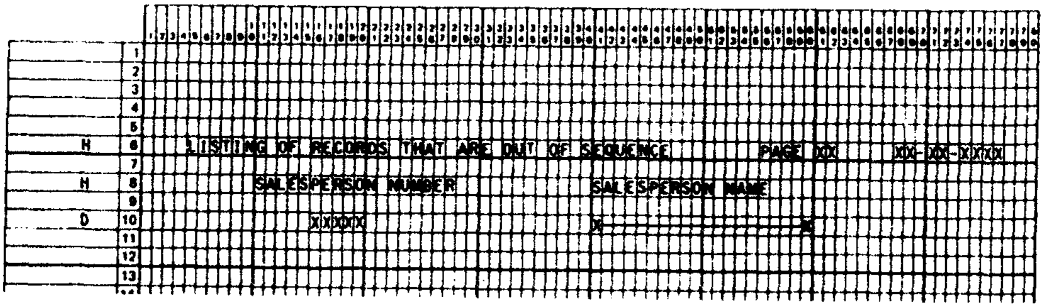


图 8.6　第 1 题的问题描述

2. 编程检查工资单记录中的错误数据其问题描述见图8.6 .

要求:

a. 执行确认程序以确保.

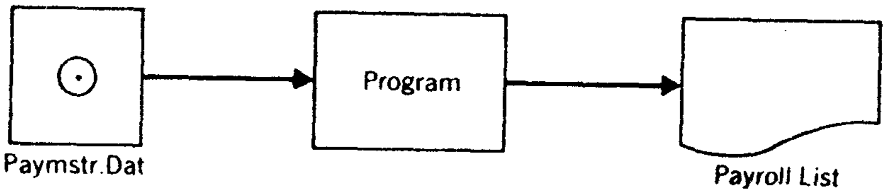
① 除EmployeeName之外的所有字段均为数字型.

② EmployeeName 字段不可省。

③ Annual Salary （每年工资）不大于 $ 125000.

b. 标题中包括日期和页号.

系统流程图

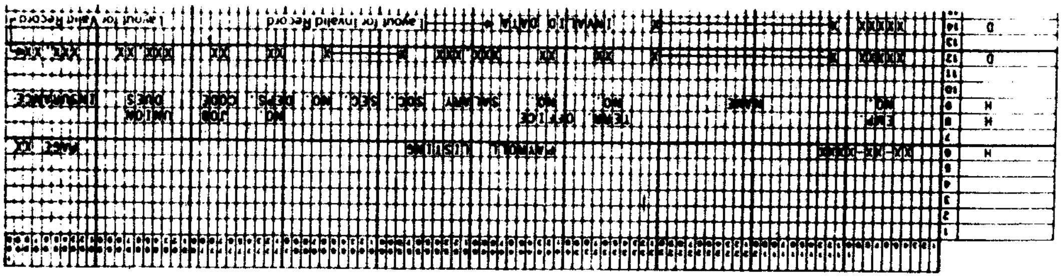


B26

Paymstr.Dat 的记录格式

EmpNo, EmpName, TerritoryNo, OfficeNo, AnnSal, SocSecNo, NoOfDependents, JobClassCode, UnionDues, Insurance

工资单打印格式图



注意：假定一行打印 100 个字符。

图 8.7　第 2 题的问题描述

3. 修改第七章的编程题4，对各输入记录进行如下确认检查：
   a. 地区号，区域号和部门号应为数字。
   b. 部门号应在1−10之间，地区号应在1−3之间，区域号应在1−3之间。
   c. 不处理含无效数据的记录，但统计无效记录的个数。
   d. 在报告的末尾打印信息：

   "THE NUMBER OF RECORDS NOT PROCESSED WAS"

   其后为所发现的无效记录个数。
4. 雇员工作时间卡包插：雇员姓名，社会救济金号，工作小时数和工作班号。确认如下内容：
   a. 除雇员姓名之外的所有字段均为数字型。
   b. 雇员姓名不可省。
   c. 工作班号为：

1	日班
2	夜班
3	假日

如果其中的数据条目出错，则该记录打印带相应错误信息。统计有效记录和无效记录的个数，并打印总数。
数据确认之后，计算应得的工资。雇员工资按如下标准付给：

日班	$7.50／小时
夜班	$12.00／小时
假日	$15.00／小时

打印各雇员所挣工资的报表。

第九章　深入了解 Turbo Basic 环境

本章要求:

1. 详细了解 Turbo Basic 环境。
2. 熟悉为满足用户需要而用来配置 Turbo Basic 环境的命令。
3. 掌握高级文本编辑功能。

§ 9.1　Turbo Basic 环境概述

注意，我们建议微机工作站的用户阅读本章，以便操作其中描述的选项和命令。

§ 9.1.1　帮助功能

如第二章所述，在使用 Turbo Basic 过程中的任意时刻，按功能键 F1 便可在屏幕上得到相应的 Help（帮助）信息。这个帮助功能是与上下文有关的（context-sensitive），即屏幕显示上 Help 信息取决于当前执行的选项或命令。这就是说，如果当前正处于 Edit（编辑）方式下（在 Edit 命令中），并按功能键 F1，那么将得到如何输入程序的帮助信息；若按 F1 时正处于 File 命令中，屏幕上则显示不同的 Help 信息。

按两次 F1 显示 Help 选项的索引

如果按一次 F1 进入 Help 功能，然后再按一次 F1，那么 Help 选项的索引将出现，如图 9.1 所示(见下页)。

此时，可以先用光标键增强用户想要的任意特定帮助选项的亮度。为选取那个选项，再按 Enter（回车）键。然后，与选定的选项相对应的 Help 信息将出现在监视器上。

正如我们已经看到的一样，主屏幕（Main Screen）由四个窗口和八个基本命令：File，Edit，Run，Compile，Options，Setup，Window 和 Debug 所组成。本章将深入讨论这些命令。

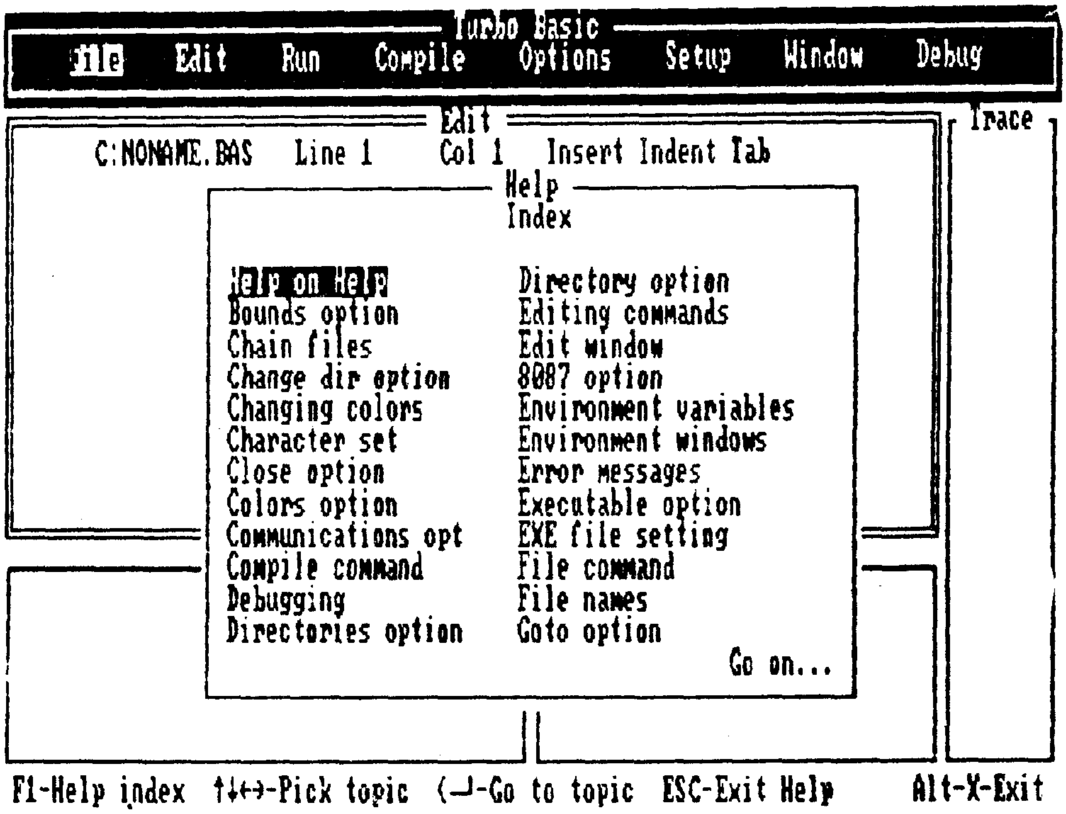


图 9.1 Help 选项的索引

§ 9.1.2 选取命令和选项

在主选单中，为选取任意的命令和选项，可以：

1. 先用光标键加亮期望的命令，按Enter键，然后再加亮期望的选项并按Enter键。

或：

2. 先按所期望命令的首字母，然后再按所期望选项的首字母。

例如，为选取 File 命令中的“Write to”选项，可以：

1. 用光标键→或←加亮File命令，然后按Enter键。接着，按三次光标键↓加亮“Write to”选项，最后再按 Enter 键。

或：

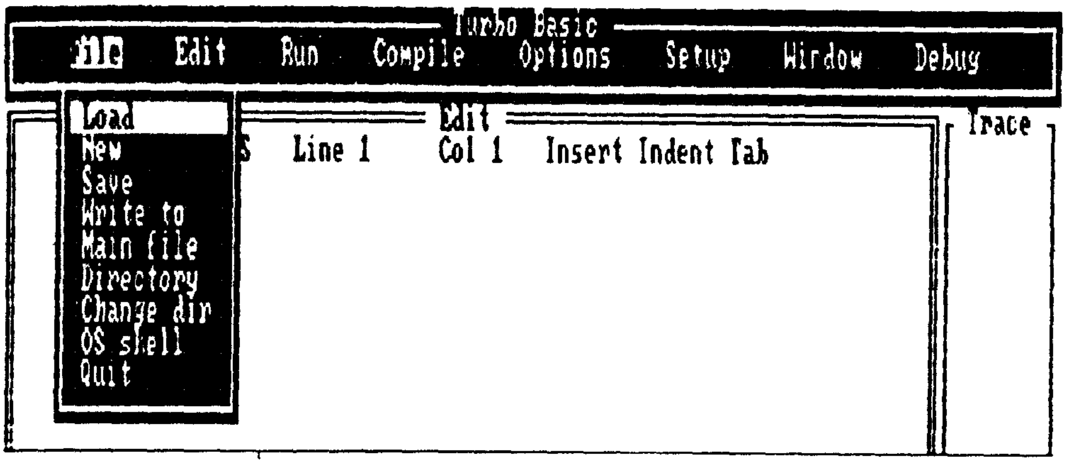
2. 敲 FW（F 对应 File 命令，W 对应于“Write to”选项）。

初级 Turbo Basic 程序员通常认为使用光标键更容易。然而，随着实际选项操作经验的增多，他们将发现键入命令和选项首字母的方法非常快。

在任意时刻，为了放弃或取消任何命令选项，按 Esc（或 Escape）键即可。假设用户已经敲了 FW，但想退回 Edit 方式下，而不是存写文件，那么按一次 Esc 键退出 Write to 选项，再按一次 Esc 键退出 File 选项。然后，既可敲一个 E，也可以先加亮 Edit 命令再按 Enter 键，返回 Edit 方式下。

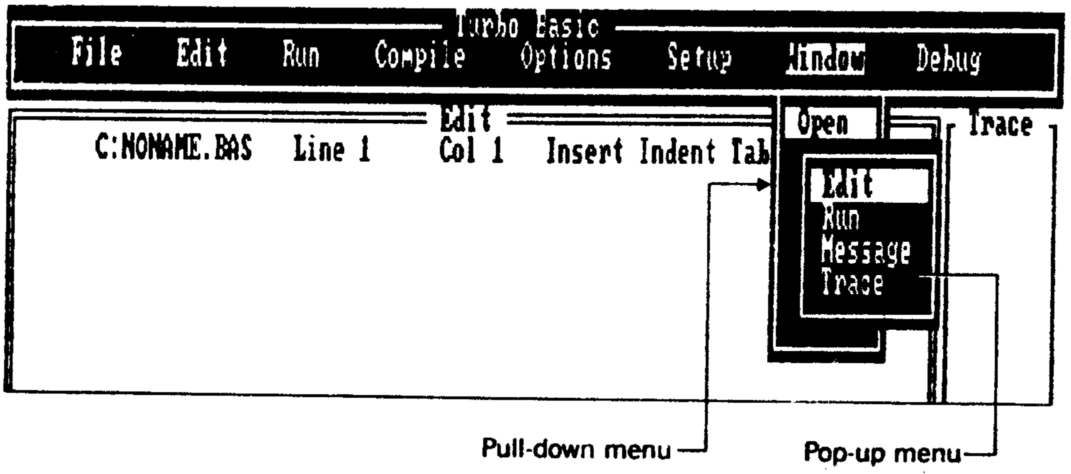
§9.1.3 下拉选单和上托选单

正如我们已经看到的一样，有许多命令在它被选时将出现一个含多个选项的子选单(Submenu)。例如，选择 File 命令时，将得到如下描述可能被选选项的子选单：



第一级子选单被称为下拉选单 (pull-down menu)。从下拉选单中选取选项时，先用↓或↑光标键加亮期望的条目，然后按 Enter 键作出选择。或者，用另一种方法，键入下拉选单中期望选项的首字母。

许多下拉选单选项还有第二级子选单。例如，选择 Window 命令中的 Open 选单时，将出现下述子选单层次：



Window 命令中 Open 选项的子选单被称作"上托选单 (pop-up menu)"，其中列出 Edit，Run，Message 和 Trace 等选项。上托选单通常叠加在下拉选单之上。同样，从上托选单中选取选项时，先用↓或↑光标键加亮相应的条目，然后按 Enter 键；或者，用另一种方法，键入相应条目的首字母。(即，上述上托选单的 E，R，M 或 T)。

如前所述，从任意选项或命令中退出时，只需简单地按 Esc 键即可。例如，要放弃对 Window 的修改，应先按一次 Esc 键清除上托选单，然后再按一次 Esc 清除下拉选单。为访问任何其他命令，使用→或←光标键，并按 Enter；或者，采用另一种方法，按该命令的首字母。

假设当前正处于 Window 命令之中，但没有按 Enter，那么不显示 Window 命令的下拉选单。为回到 Edit 方式下，用光标键加亮单词 Edit，并按 Enter；或在主选单中简单地按 Edit 命令的首字母 E。

§9.1.4 确定运行状态

回忆一下，通常有四个窗口，(Edit，Message，Trace 和 Run) 出现在屏幕上。注意，带双线边框的窗口为当前窗口 (active window)。假设用户选择了 File 命令下的 Save 选项，但未记住是否在存盘之前已编译过该程序。如果 Edit 窗口有双线边框，那么它就是当前窗口；因此，存盘之前，用户正处于 Edit 方式下。另一方面，如果 Message 窗口为当前窗口，那么用户必定事先编译过这个程序。否则，若 Run 或 Trace 窗口为当前窗口，那么同样知道，用户已事先执行过该程序。

§9.2 重新配置屏幕

到目前为止，我们用缺省配置的窗口描绘过 Turbo Basic 环境。如果用户喜欢，可以重新配置 Turbo Basic 环境，以便各窗口占用不同的体积、配置不同的颜色，以及出现在屏幕上不同的地方。

§9.2.1 Tile 选项

Window（窗口）命令下的 Tile（平分）选项使所有四个窗口均显示成同样大小（参见图 9.2）。

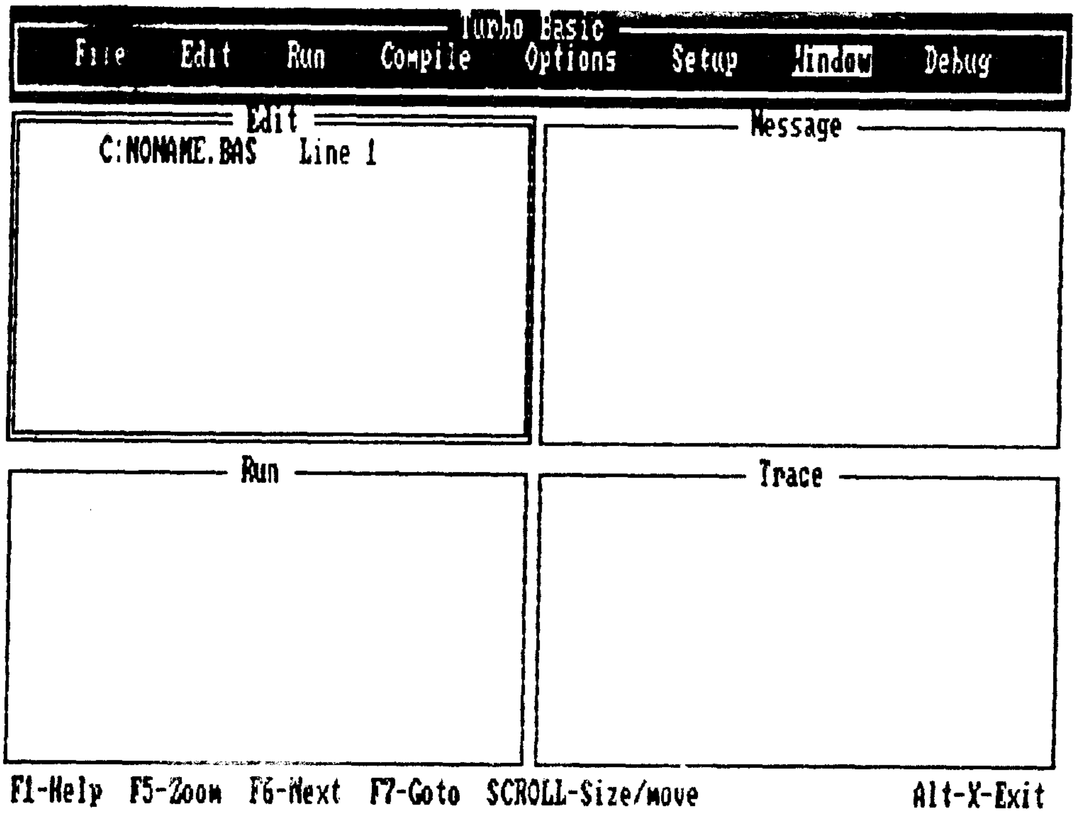
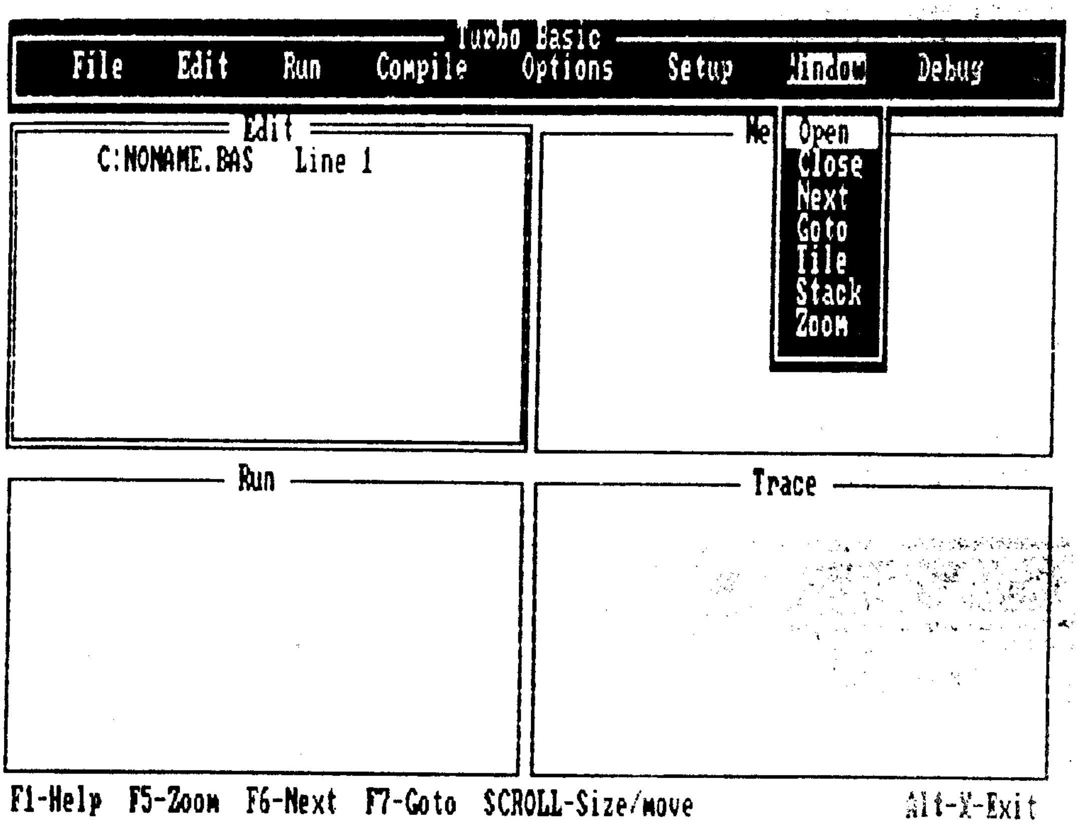


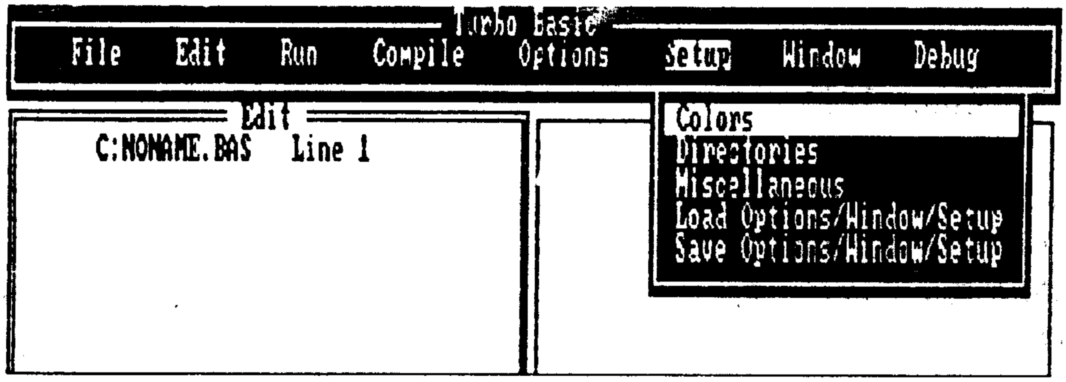
图 9.2 平分后的屏幕显示

为了将这四个窗口显示成大小相等的盒子，先用光标键→加亮 Window 命令，然后再按 Enter 键，或在命令模式下按字母 W。下述下拉选单将出现：

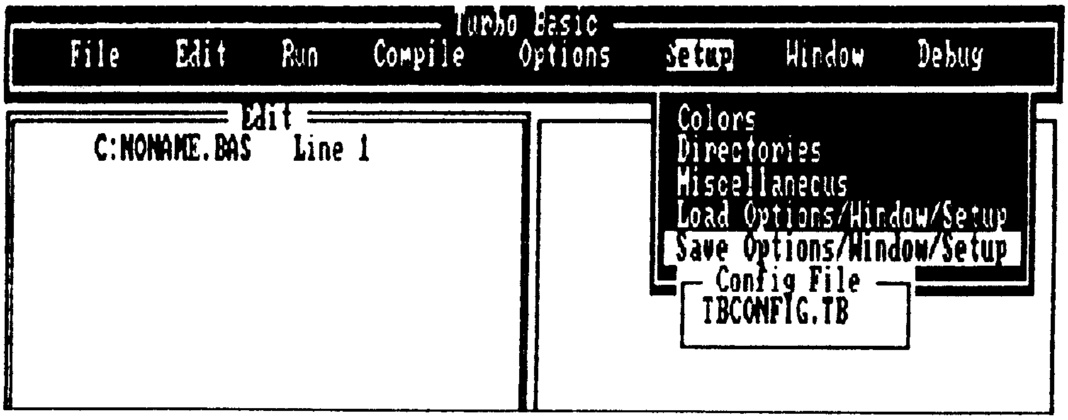


先按四次下光标键↓加亮 Tile 选项，再按 Enter（或敲 Tile 的首字母 T）；然后按 Esc 键退出 Window 方式。这样，所有四个窗口将显示成大小相等的盒子。

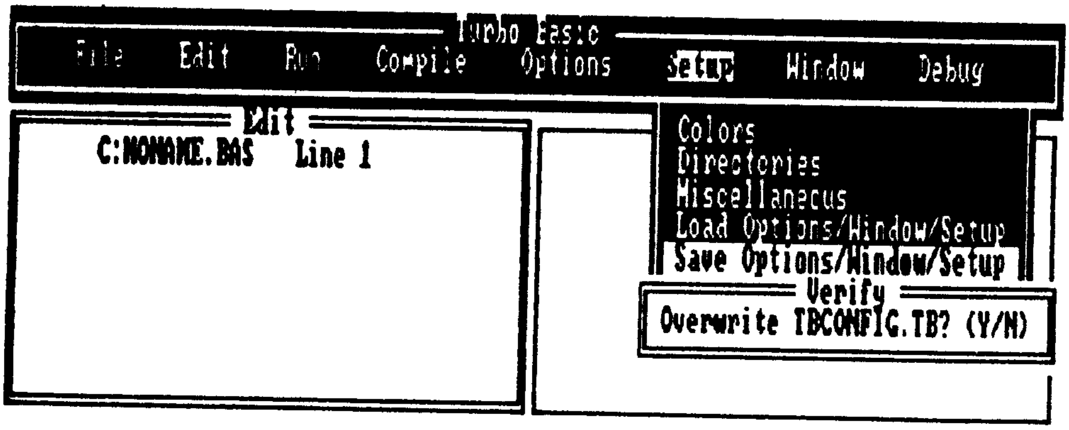
为了永久保存这个 Tile 选择，以便使它成为新的缺省值，用光标键→加亮 Setup 命令，并按 Enter 键。或者，在命令模式下，敲 Setup 命令的首字母 S。此时，将出现如下屏幕显示：



接着，按四次下光标键↓加亮 Save Options／Window／Setup 选项，并按 Enter 键（或敲 Save 的首字母 S）。下述屏幕显示将出现：



该选项给缺省的配置文件命名；该文件保存所作出的修改。按 Enter 键时，用户被要求确认是否希望作出永久性的修改。



如果按 Y 确认，那么这个平分的、大小相等的屏幕将存入系统缺省配置文件 TBCONFIG·TB。启动 Turbo Basic 时，Turbo Basic 系统自动调用该系统配置文件。这就是说，今后将它装入 Turbo Basic 系统之中时，屏幕显示将由四个大小相等或平分的窗口组成。

§9.2.2 Zoom 功能

Tile 选项是将所有窗口显示成大小相等的盒子的一种简便方法。不过，请记住，用户可使用 Zoom（放缩）功能仅在 Edit 窗口中显示的程序覆盖整个屏幕。同样，也可以使用 ZOOM 功能使 Run 窗口覆盖整个屏幕。Zoom 功能覆盖的是 Run 窗口还是 Edit 窗口，仅取决于口哪个窗口是当前窗口，而不管它是否被平分。

当前窗口	Zoom 的方法
Edit 窗口	按 F5 键
Run 窗口(程序未执行时)	按 F5 键
Run 窗口(程序已执行时)	按 Alt–F5 键

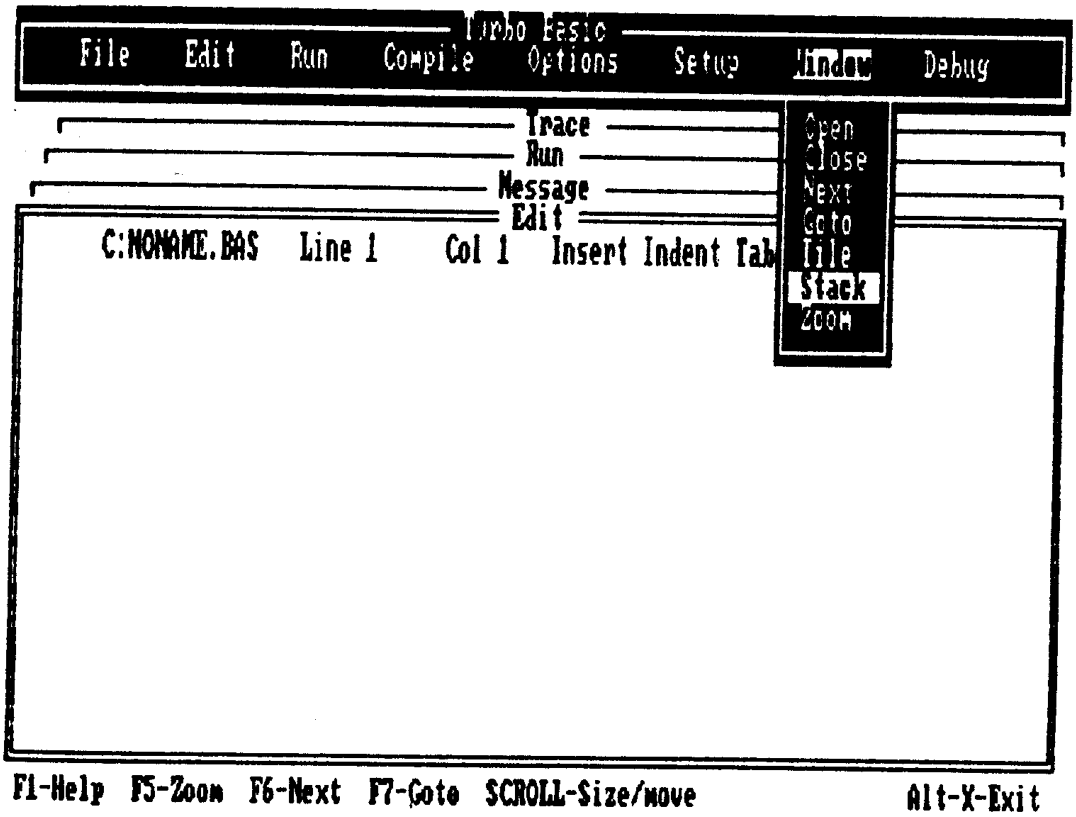
为回到缺省的窗口，在 Edit 方式下再次按 F5 键，或在 Run 方式下再按一次 Alt−F5。这就是说，这些键均为乒乓开关键。

只有 Run 和 Edit 窗口可放缩。此外，为了放缩，相应的 Run 窗口或 Edit 窗口必须为当前窗口。

§ 9.2.3 Stack 选项

Window 命令的另一个选项被称作“堆积（Stack）”选项。Stack 选项使所有的窗口相互覆盖，一个窗口出现在另一个窗口的“顶部”或前面。选用 Stack 选项时，当前窗口位于顶层。当前窗口就是在选用 Window 命令之前激活的那一个窗口。如前所述，当前窗口用双线边框标识。假设当前处于 Edit 命令方式下，并希望堆积窗口，那么先按 Esc 键退出 Edit 方式，然后按 W 执行 Window 命令。

或者，采用另一种方法，先用光标键→加亮 Window 命令，然后再按 Enter 键。在两种情况下，Window 命令的下拉选单选项都将出现。如果接着按五次下光标键↓加亮 Stack 选项，并按 Enter 键，屏幕上将出现：



注意：四个窗口被堆积，Edit 或当前窗口在最上层，随后依次为 Message，Run 和 Trace 窗口。每按一次功能键 F6，堆中的下一个窗口都将前卷。

窗口堆积类似于有一堆单独的纸张，其中一张在另一张的上面，每张均可分别翻阅。

为清除 Window 命令的下拉选单时，按 Esc 键。

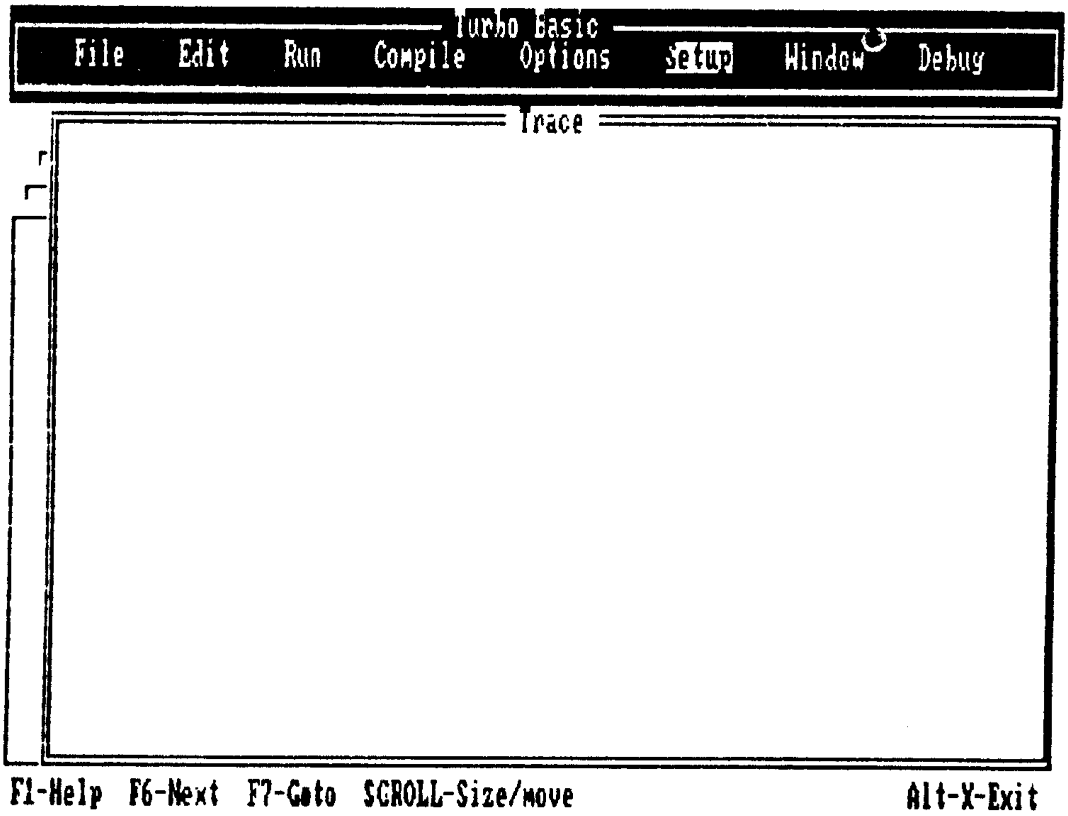
注意，在上述图例中，Edit 窗口位于最上层的原因是因为它是当前窗口。也就是说，因为在选用 Stack 选项之前我们处于 Edit 方式下，因而 Edit 窗口覆盖其他窗口。换一种情况，如果我们处于 Run 方式下，那么 Run 窗口将出现在最上层。还应注意，当前窗口总是用双线框住，而非当前窗口的框线为单线。

§9.2.4 改变当前窗口

§9.2.4.1 使用功能键 F6

窗口被堆积时，按功能键 F6 覆盖堆中最后一个窗口，以便使之出现在最上层，并成为当前窗口。无论什么时候，只要 F6–Next 出现在屏幕的底部，即 Help 行，功能键 F6 便可在堆积窗口上产生这个结果。用户将发现 F6–Next 在大多数屏幕画面上出现。处理堆积窗口时，提示符——F6–Next——提醒用户为进入堆中下一个窗口必须按功能键 F6。每按一次 F6，堆中最后一个窗口将提到顶层，与翻书相似。在 Run 方式下，Help 行显示"Alt–F6"；它表示在这种方式必须使用 Alt–F6，而不是 F6 本身。

假设 Edit 窗口是堆积屏幕上的当前窗口。如果按一次功能键 F6，那么 Trace 窗口变成当前窗口，并覆盖其他三个窗口（参见下图）。



请记住，按一次 F6 时，堆中最后一个窗口变成最上层窗口，同时也成为当前窗口。

再按一次 F6 将 Run 窗口提到顶层，并使它成为当前窗口。如果再按一次 F6，那么 Message 窗口将随之成为最上层窗口。接着按一次 F6 将使 Edit 屏幕作为当前窗口回到堆顶。

因此，堆积窗口可使用户为各窗口使用整个屏幕。为依次访问和激活各窗口，只要简单地按功能键 F6 即可。

窗口显示的缺少顺序为：Edit 后跟 Trace，Trace 后跟 Run，Run 后跟 Message。

F6 也可以用来改变平分或缺省屏幕上的当前窗口。假如 Edit 窗口为当前窗口，那么每按一次 F6，当前窗口依次由 Edit 变为 Trace、Run、Message，并再次回到 Edit，如此继续。

§ 9.2.4.2 Window 命令中的 Next 选项

据前面所述，在大多数情况下，通过使用功能键 F6 便可改变当前窗口。另一种方法是，使用 Window 命令下的 Next（下一个）选项使指定窗口成为当前窗口（参见图 9.3）。

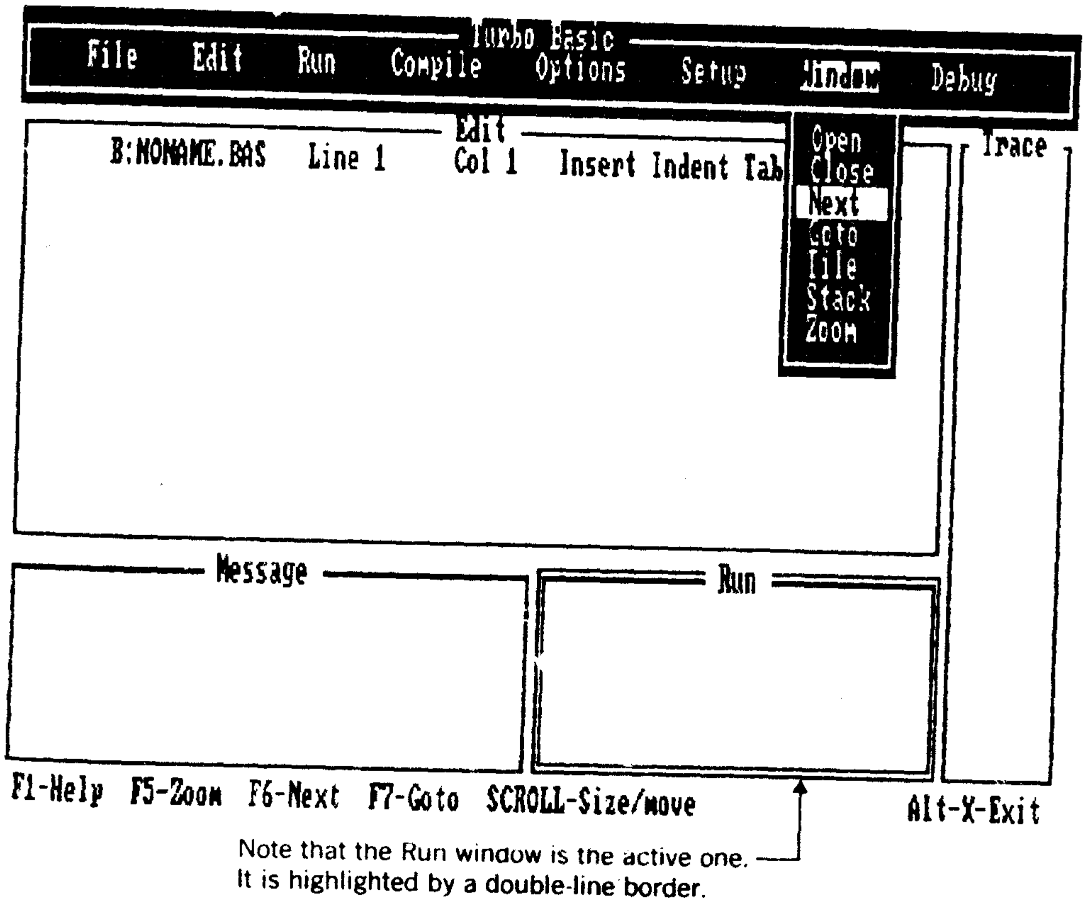


图 9.3 Next 选项

不管当前处于那个命令层次，该选项均可工作。假如选作 Window 命令下的 Next 选项，那么每按一次 Enter 键，四个窗口中的一个不同的窗口将成为当前窗口。这就是说，该选项起的作用与 Help 行中出现 F6 时功能键 F6 所起的作用相同。

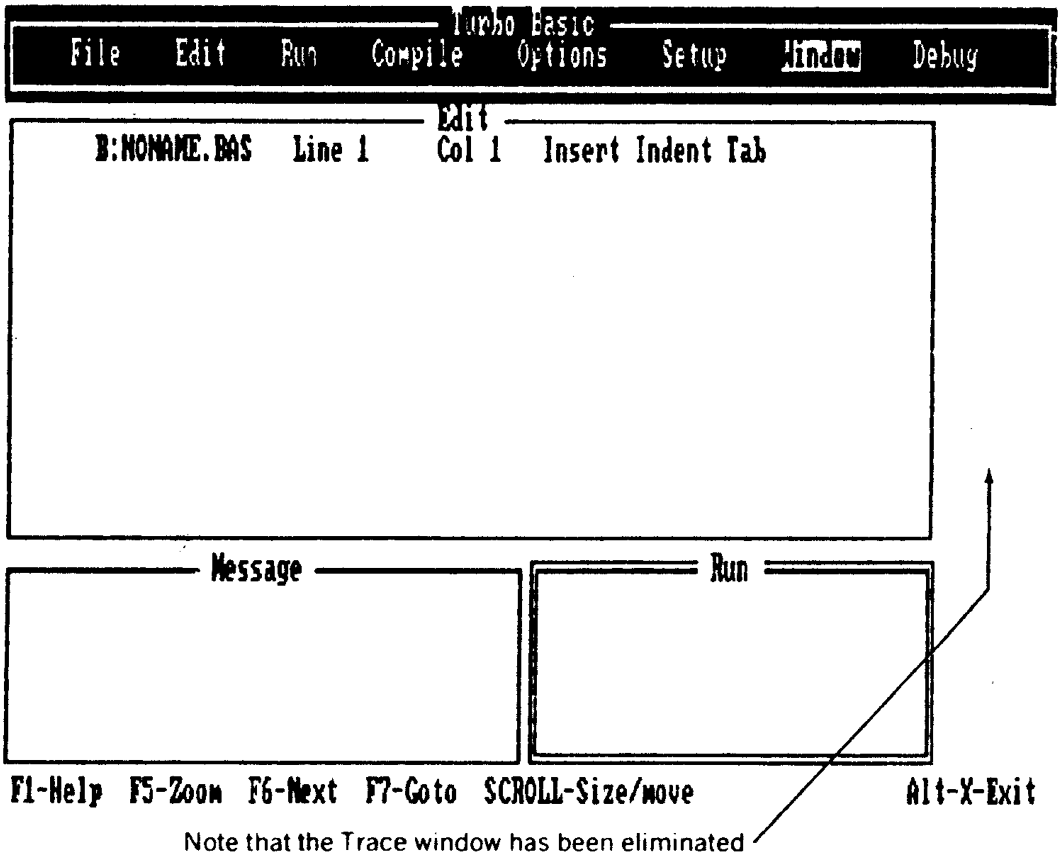
如前所述，Turbo Basic 通过使用双线边框的方法突出当前窗口；所有其他窗口的边框均为单线框。象 F6 键一样，Window 命令下的 Next 选项可用来激活任何窗口，而不管它是平分的、堆积的，还是用缺省方法显示的。

§9.2.5 将窗口从屏幕上清除

在某些情况下，用户可能想永久地或为特定的应用程序删除或清除一个 Edit，Run，Trace 或 Message 窗口。为了清除一个窗口，必须先使之成为当前窗口，然后再将它关闭。

窗口既可从缺省设置的屏幕、平分的屏幕上，也可从堆积的屏幕上清除。为将窗口清除，应先用 Window 命令下的 Next 选项或按所需次数的 F6 键使之成为当前窗口。然后再用光标键↑加亮 Window 命令下的 Close 选项，并按 Enter 键。这样，刚激活的窗口将不再出现在屏幕。

例如，为从缺省屏幕上清除 Trace 窗口，先选取 Window 命令下的 Next 选项，并按 Enter 键直至 Trace 窗口具有双线边框为止，使之激活，成为当前窗口。然后再选取 Window 命令下的 Close 选项。此时有如下画面：

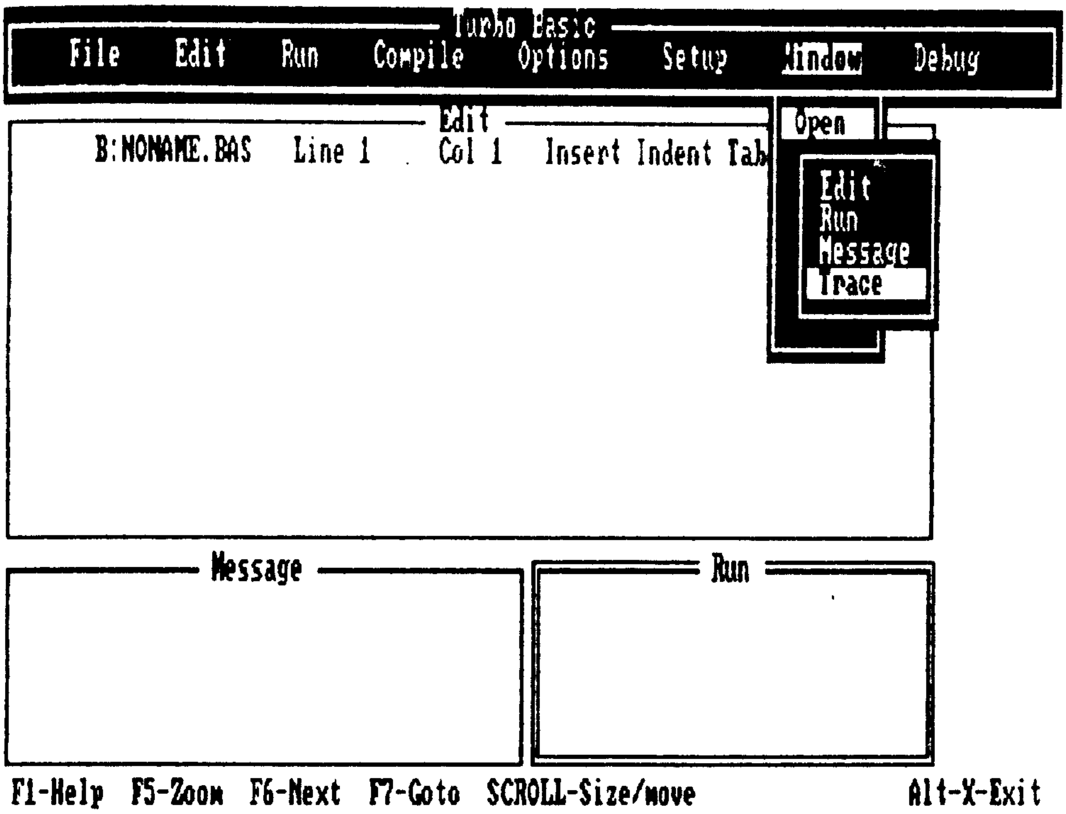


用这种方法可将屏幕上的任何窗口清除。

如前所述，用户可以重新配置屏幕形式，并使所作出的改变永久化，成为缺省的设置。为此，先作出修改，然后加亮 Setup 命令下的 Save Options / Window / Setup 选项；接着按两次 Enter 键，并按 Y 响应要求确认存盘与否的提示。

§ 9.2.6 在屏幕上恢复已关闭的窗口

为重新调用被清除的窗口，使用 Window 命令下的 Option 选项。假设如上图所示事先已清除了 Trace 窗口，并希望使之恢复，那么先加亮 Window 命令下的 Open 选项并按 Enter 键。此时，下述上托选单将出现



如果在 Tuace 选项被加亮时按 Enter 键，Trace 窗口将被恢复。

总之，先用 Window 命令下的 Next 选项，或在 Help 行出现 F6–Next 时按 F6 键，使一个窗口成为当前窗口；然后再用 Close 选项消除当前窗口。最后，可用下述方法恢复关闭的窗口：先加亮 Open 选项，按 Enter 键，选择要激活的窗口，然后再次按动 Enter 键最后按两次 Esc 键删除下拉选单和上托选单。

下面，我们将讨论改变窗口大小、重新定位和给屏幕增加颜色等方法。

§ 9.3 进一步讨论 Window 命令

§ 9.3.1 激活窗口

有很多方法可激活窗口。在 Edit 模式下，Edit 窗口是当前窗口；同样，在 Run 模式下，Run 窗口是当前窗口。

为改变当前窗口，选择 Window 命令，并使用下述任一选项:

1. 先选取Next选项（加亮Next选项并按Enter键，或敲N)，然后每按一次Enter键，下一个窗口依次成为当前窗口。如前所述，其顺序为 Edit，Trace，Run 和 Message。
2. 先选取Open选项（加亮Open选项并按Enter键，或敲O)，然后从上托选单中选取要使之激活的窗口。

另一种方法是，如果"F6—Next"出现 Help 行上，重复按功能键 F6，直至期望的窗口成为当前窗口为止。

§ 9.3.2 Window 命令的 Goto 选项

不必执行其他命令，使用 Goto 选项便可回到当前窗口。

例如，假设当前处于 Edit 模式下并决定清除 Message 窗口，那么先按 Esc 键，选取 Window 命令下的 Open 选项，然后再选取 Messgae 选项。为清除 Message 窗口，选取 Close 选项．此时，当前窗口再次为 Edit 窗口。为回到 Edit 方式下，可用如下两种方法:

(1) 按 Esc 键并选 Edit 命令

(2) 简单地选取 Window 命令下的 Goto 选项，它可立即回到 Edit 方式下。

此外，还可以用另一种替代方法：如果 Help 行上出现"F7—Goto"，按功能键 F7 便同样立即回到 Edit 方式下。

§ 9.3.3 Window 选项小结

使用 Window 选项可永久地或为特定的应用程序重新配置屏幕。

Window 选项摘录如下:

选项	意　义
Open	使指定的窗口成为当前窗口或恢复已消除或删除的窗口。将激活的窗口从上托选单中选取。
Close	清除当前激活的窗口，命名之不再出现在屏幕上。
Next	使下一个窗口依次激活（顺序为 Edit, Trace, Run, Messgae)。将激活窗口的选取方法为：重复按 Enter，直至期望的窗口具有双线边框时为止。
Goto	不必回到命令模式下，直接返回当前窗口。
Tile	将屏幕细分为四个大小相等的窗口（如果在平分之前关闭了一个或多个窗口，它们可从屏幕上删掉)。
Stack	重叠或堆积所有的窗口，使各窗口均占用整个屏幕，好像它们是四页纸，相互重叠（通过关闭的方法，可删除堆中单个或多个窗口)。
Zoom	对 Run 或 Edit 窗口进行缩放；具体窗口取决于哪个为当时的当前窗口（另一种方法是：在 Edit 方式下，按 F5 键缩放 Edit 窗口；或在 Run 方式下，按 Alt-F5 键缩放 Run 窗口)。

上述 Window 选项出现在 Window 命令的下拉选单中。如果在 Window 选项中选取 Open 或 Close，那么列出四个窗口的子选单将出现。该子选单被称为"上托选单"。

自我测验(一)

1. 为显示帮助信息，按功能键____。
2. （T 或 F）帮助信息总是一样的。
3. 为选取File命令下的Load选项，可使用光标键加亮期望的命令和选项，或更简单地在主选单中按____。

4. 选取 File 命令时，出现在屏幕上的选单叫做____.
5. 选取 Window 命令下的 Open 选项时，出现在屏幕上的选单叫____.
6. (T 或 F) 所有的下拉选单均有上托选单。
7. 为从任何选项或命令中退出，按____键。
8. 在 Turbo Basic 中，双线框的窗口被称作____窗口.
9. 为显示四个大小相等的窗口，选取____命令下的____选项.
10. (T 或 F) 只有 Run 和 Trace 窗口可缩放。
11. (T 或 F) 平分后的 Turbo Basic 窗口可设为缺省窗口.
12. (T 或 F) 为从屏幕的显示中消除窗口，该窗口必须为当前窗口.
13. (T 或 F) 为从屏幕的显示中消除当前窗口，选取____命令和____选项.
14. (T或F) 使一个窗口成为当前窗口的一种方法是：先选取Window命令下的 Next 选项，然后一直按 Enter 键，直到双线框围绕期望的窗口为止.
15. (T或F) 使一个窗口成为当前窗口的一种方法是：先选取Window命令下的 Goto 选项，然后选取要激活的具体窗口。

§9.4 Turbo Basic 屏幕底部 Help 行的解释

§9.4.1 在命令选取之前出现在 Turbo Basic 主屏幕上的提示

刚开始装入 Turbo Basic 系统时，出现初始屏幕．按加车键，随即出现主屏幕，其中 File 命令被加亮．在屏幕底部出现如下提示：

```
F1-Help F5-Zoom F6-Next F7-Goto SCROLL-Size/move        Alt-X-Exit
```

这行被称做"帮助行 (Help Line)"．它包括有关键及其功能．根据当前被执行命令的不同，Help 行发生变化．上述帮助表示：

1. 如果按F1，便可得到与上下文相关的帮助．这意味着，帮助的形式取决于当前正使用的命令或选项．例如，若当前处于 Edit 命令下，按 F1 键时则可得到解释 Edit 特性的帮助信息；若当前处于 Run 命令下，则出现 Run 命令的解释；等等.
2. F6——与Window命令下Next选项所执行的操作相同．即，每按一次F6，下一个窗口将依次成为当前窗口.
3. F7——与Window命令下的Goto选项所执行的操作相同．即，不必进入有关命令下，可直接回到当前窗口中。
4. 通常位于键盘右上部分的Scroll Lock键可用于修改窗口的体积和移动窗口（详见后面的解释).
5. 如果按Alt键，并在按住Alt键时按X键，则脱离Turbo Basic环境，并回到DOS状

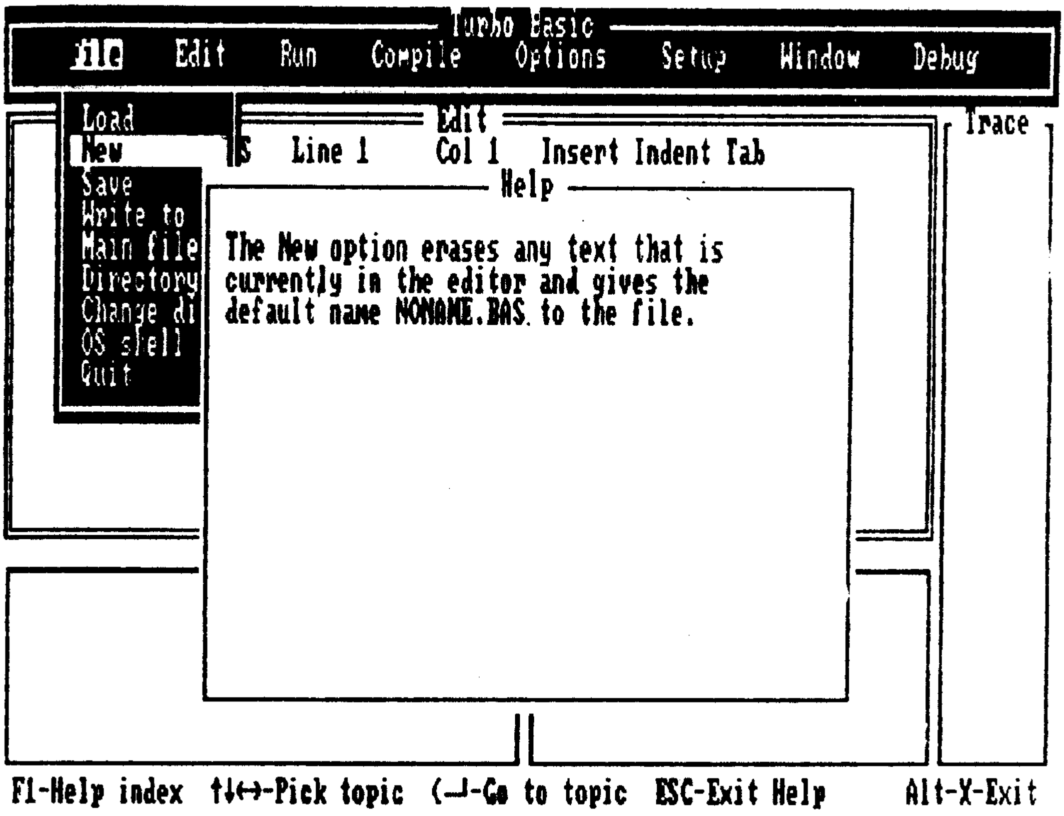
态下．按 Alt−X 后，若要重新进入 Turbo Basic，在 DOS 提示符敲入 EXIT，然后再敲 TB．

如果 CAPS 重叠出现在 Alt−X 上面，那么触发了 CapsLock 键，所有字母均为大写字母．如果叠在上面的是 NUM，那么触发了 NumLock 键，数字小键盘上的光标键均失效；除非 NumLock 键被再次触发，否则只有其中的数字才可传入计算机．

§9.4.2 帮助行随所执行命令的不同而变化

如同 Help 功能本身一样，Help 行也是与上下文有关的，其中显示的提示随所执行命令的不同而不同．因为 F1 键总是提供帮助，因而“F1−Help”出现在各 Help 行的左边；而不管哪个命令正被执行．同样，不管当前正执行的 Turbo Basic 命令是什么，Alt−x 总用来退出 Turbo Basic．但是，帮助行其余部分将随所执行命令的不同而显示不同的键及其功能．因为我们已讨论过大多数被显示键的用处，故它们的功能对用户来说比较清楚．

下面讨论一个例子：先加亮 File 命令下的 New 选项，然后按功能键 F1，则有如下屏幕显示：



总之，用户可以在任何命令或选项下按功能键 F1．查看上述屏幕底行，可以注意到其意义如下：

Help 行(底行)	意　义
F1 － Help index	在 Help 功能下，再按 F1 可得到帮助选项的索引.
↓ ↑ ←→ Pick topic	用光标键加亮任何帮助条目
↵ Go to topic	按 Enter 键选取光标处帮助条目.
Esc － Exit Help	按 Esc 退出 Help 功能
Alt-X － Exit	按 Alt-x 退出 Turbo Basic

§9.4.3　热键

Turbo Basic 有一系列热键（hot key）；不管当前位于主选单中的什么地方，热键均具有相同的意义．例如，如上节所述，在任何地方要得到与上下文有关的帮助信息，均按 F1 键．任意时刻要想退出 Turbo Basic 系统，按 Alt-x。Alt-x 之所以叫做热键，是因为它总是产生相同的结果.

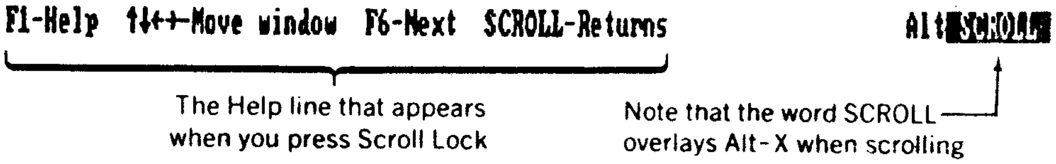
下面为 Turbo Basic 的热键

热键	功　能
F1	提供与上下文有关的帮助
F2	将文件存盘
F3	创建新的文件
F5	放缩 Run 或 Edit 窗口
F6	切换当前窗口
F7	从选单系统中进入当前窗口
Scroll Lock	修改窗口体积和移动窗口
Esc	返回主选单
Alt-C	编译当前装入的程序
Alt-E	编辑当前装入的程序
Alt-R	运行当前装入的程序
Alt-X	退出 Turbo Basic 系统
Alt-F	进入 File 选单
Alt-O	进入 Options 选单
Alt-D	进入 Debug 选单
Alt-W	进入 Window 选单

§ 9.5.　改变窗口大小

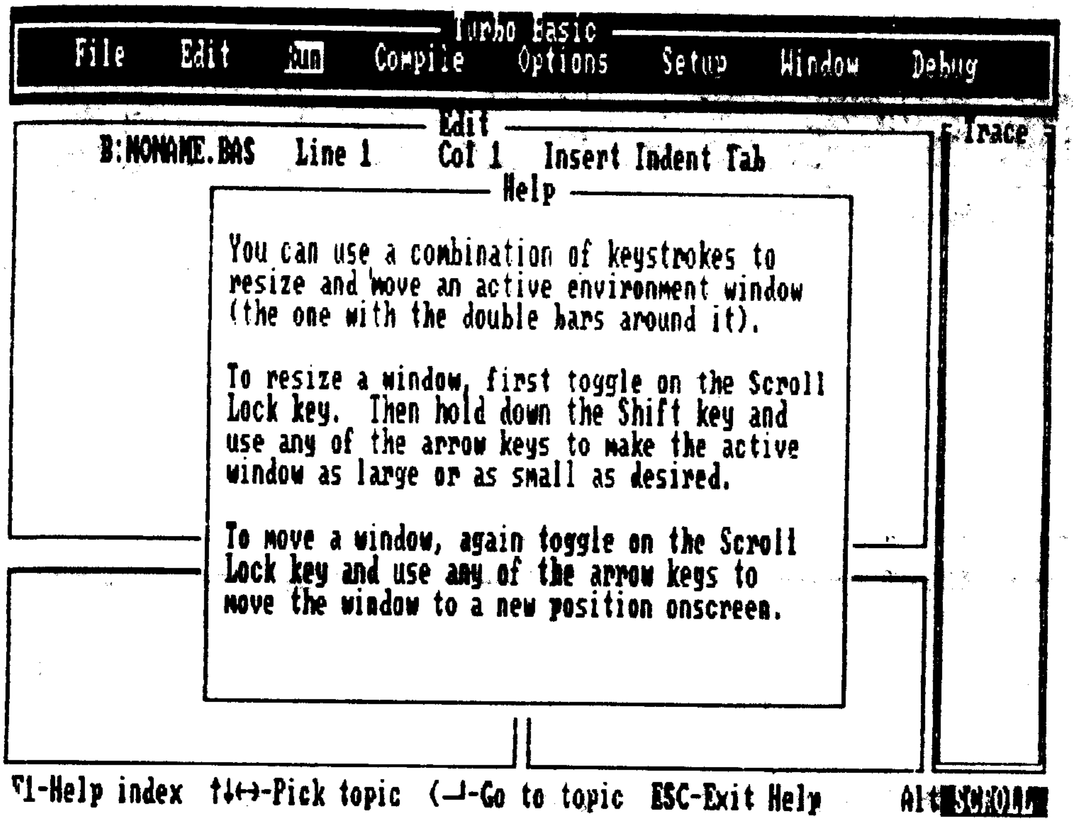
如前所述，缺省（或其他）屏幕显示的帮助行中包括如下信息：**SCROLL**—Size／move。它表明，位于大多数键盘右上部的 Scroll Lock 键可用来改变窗口的大小和移动窗口。将受 Scroll Lock 键影响的窗口为当前窗口。

假设当 Trace 窗口为当前窗口时按 Scroll Lock 键。这样就可调整 Trace 的大小，并可移动 Trace 窗口。按 Scroll Lock 将在屏幕底部的 Help 行上产生一组不同的提示:



当用户为重新定义窗口大小并确定其位置而按 Scroll Lock 时，单词 SCROLL 被加亮显示在帮助行的右边，并如上图所示迭加在 Alt—X 提示之上。这表示 Scroll（翻卷）功能被激活。

此时，按 F1 可得到如下的与上下文有关的帮助信息:



该帮助窗口表示，按住 Shift 键时按光标键可改变当前窗口的大小（在这些例子中，Trace 窗口为当前窗口）。为脱离 Help 选单，恢复前面的屏幕显示，按 Esc 键。

此时，使用 Shift 键与光标键，可重新配置任何满足用户各自需要的窗口。键盘上标有向上箭头↑的键为 Shift 键。

按 Shift 键，并将它按住时，屏幕底部的帮助行上将出现如下提示：

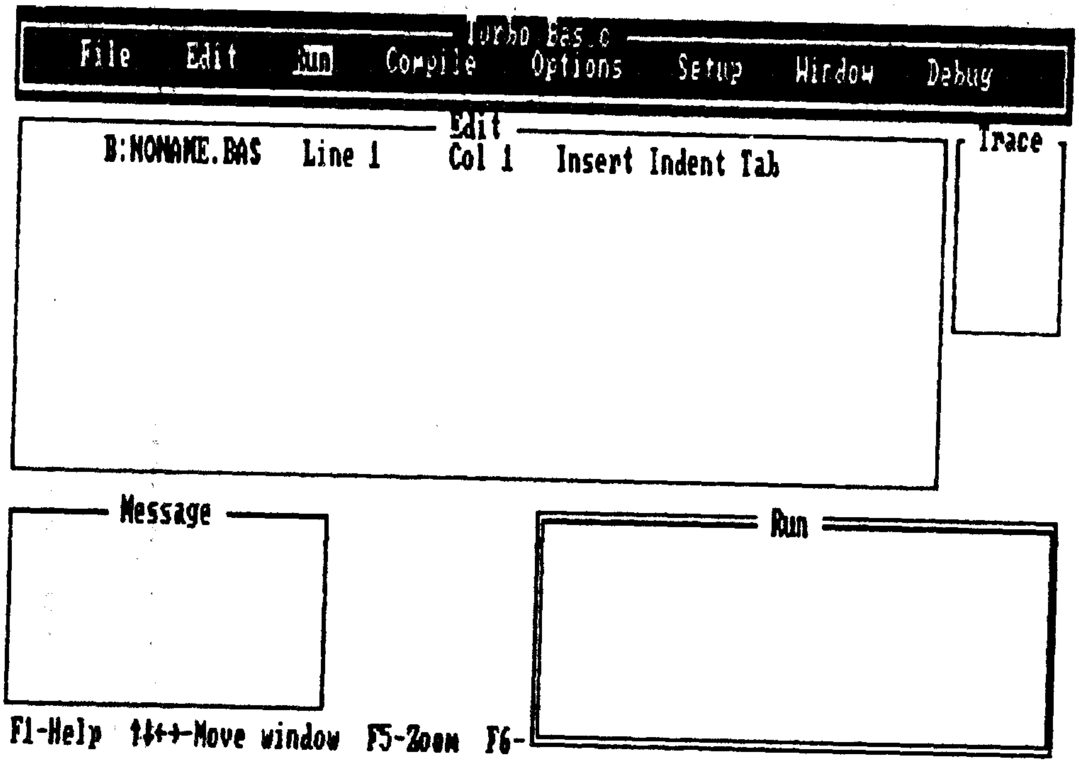
↑ ← —Contract window　　↓ → —Expand window

下述各键的作用如下(连字符“–”表示同时按键)：

键	作　用
Shift–↑	减少当前窗口的高度
Shift–←	减少当前窗口的宽度
Shift–↓	增加当前窗口的高度
Shift–→	增加当前窗口的宽度

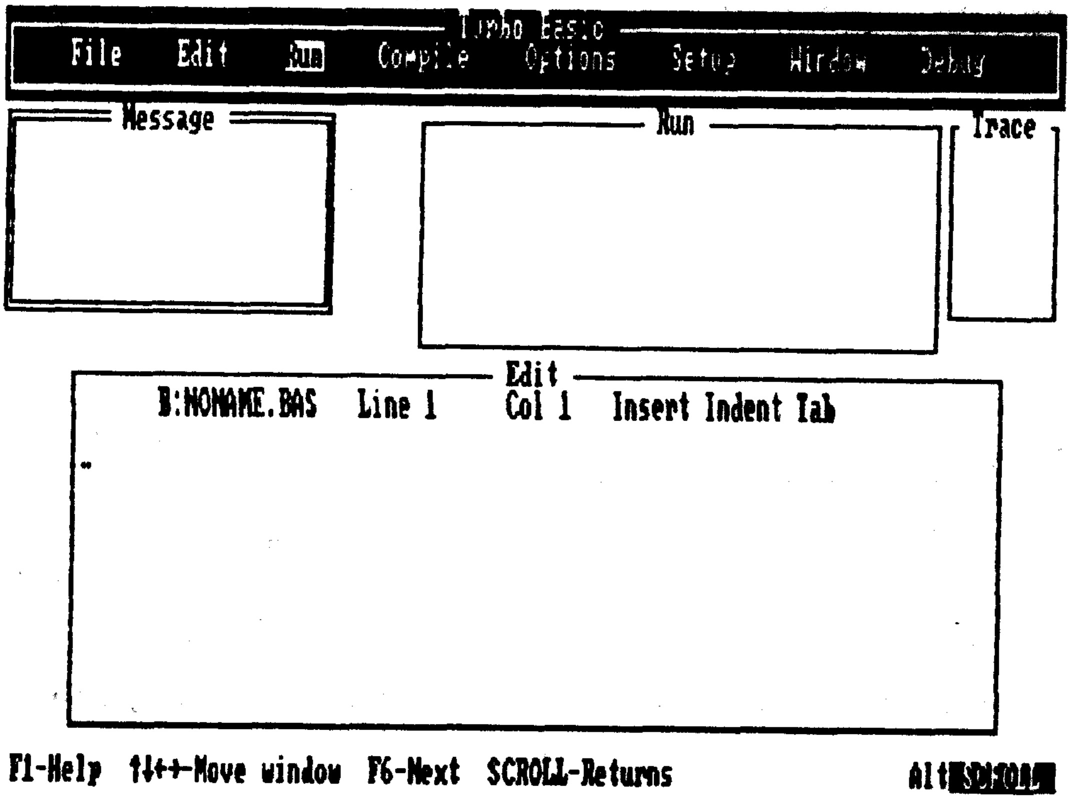
用户可以先使用 F6 键，或 Window 命令下的 Next 选项激活其他窗口（使之成为当前窗口），然后，可调整它的大小。

下面是使用 Shift 键和光标键改变窗口大小时可能显示的图例：



类似地，不用 Shift 键而使用光标键本身，可以重新将窗口定位或移动。不过，用户应确信已按 Scroll Lock 键，并且单词 SCROLL 高亮度显示在帮助行上。

下图所述的重新配置可通过先使用 Shift 键和光标键修改窗口大小，然后再用光标键本身重新将窗口定位或移位的方法获得：



屏幕（即窗口）既可移至不同的位置，又可相互重叠（见下页图 9.4）．

此时，按 Scroll 键回到其他 Turbo Basic 命令和选项中．

重新确定窗口大小和位置的小结

按 Scroll Lock 键重新确定当前窗口大小和／或位置．

1. 使用 Shift 键和光标键改变当前窗口大小．
2. 单独使用光标键可重新确定当前窗口的位置．
3. 按 Scroll 键回到命令模式．

在窗口重新确定了大小和位置，满足用户的要求之后，应确信再次按动或触发 Scroll Lock 键，以便脱离 Scroll 模式．

此外，用户也可通过选取 Setup 命令下的 Save Options／Window／Setup 选项，并将变化存入 TBCONFIG.TB 文件中的方法，保存其大小和位置被重新确定后的窗口，使之成为缺省设置．

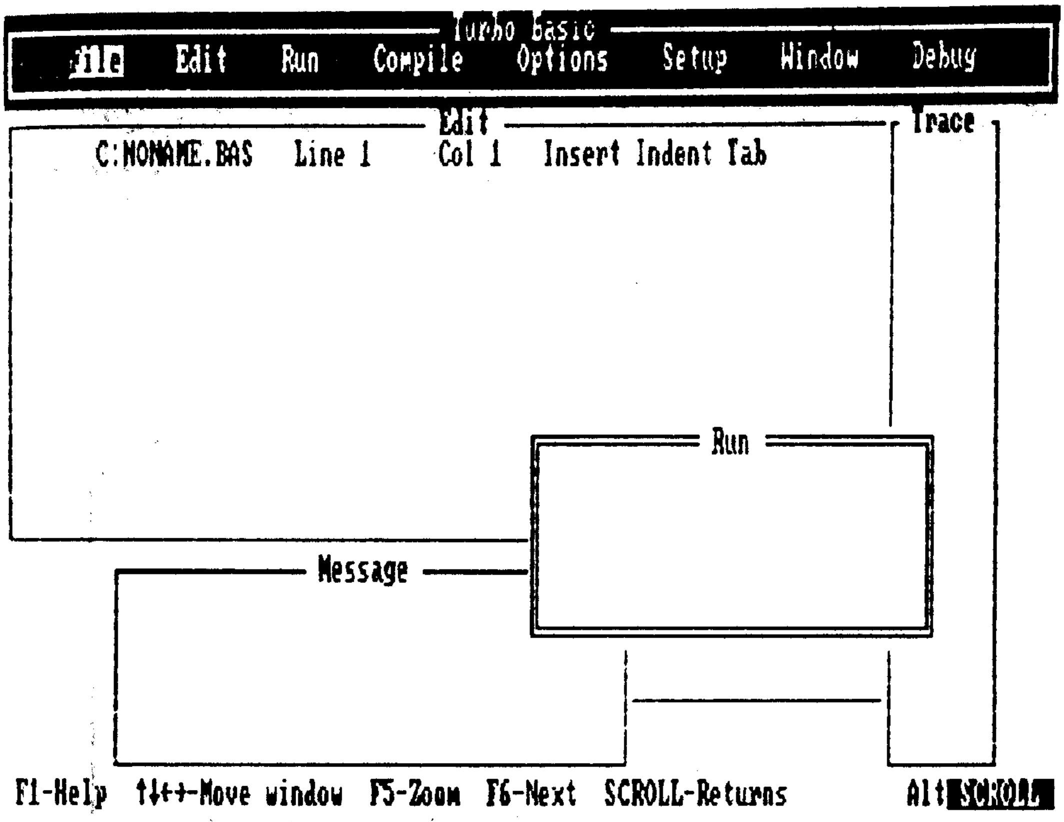


图 9.4 窗口重叠

§9.6 用不同的配置文件建立不同的环境

假如用户希望为不同的应用场合或程序设立不同的配置. TBCONFIG.TB 是缺省的配置；在 Turbo Baisc 系统装入初出现的配置总是该缺省配置.

但是，用户可为特定的应用场合设定任意数目的不同配置. 为保存指定的环境或窗口配置，使用前面介绍过的 Setup 命令下的 Save Options / Window / Setup 选项。此时，下述上托窗口出现，其中带有指向文件名的光标.

```
── Config File ──
 TBCONFIG.TB
```

如果用户键入不同的名字替换 TBCONFIG.TB，如 ALTCONFG.TB，那么指定的配置将存入文件 ALTCONFG.TB，脱离缺省的配置，且不改进 TBCONFIG.TB. 给配置文件命名时，总是用扩展名.TB。

无论什么时候要使用另一个可替换配置，都可先装入 Turbo Basic 系统，然后再选取

Setup 命令下的 Load Options / Window / Setup 选项，并键入用于该配置的文件名（在当前例子中为 ALTCONFG.TB）；接着按 Enter 时，该替换配置将出现在屏幕上。用这种方法，用户可创造若干个不同的配置，并在需要时调入。

§ 9.7 Setup 命令下的 Color 选项

如果有彩色监视器，用户可在屏幕上用不同的颜色实际显示各种元素。为彩色显示而配置屏幕时，加亮 Setup 命令下的 Color 选项。

此时，上托选单中出现如下三个条目：

```
Windows
Menus
System boxes
```

如果加亮 Window 选项并按 Enter 键，那么下述四个窗口将出现在另一个上托选单中：

```
Editor
Run
Message
Trace
```

这些窗口中的每一个均可用不同颜色显示。甚至，窗口中的各个元素，如标题、边框、正文等，均可用不同颜色显示。

同样，如果选取 Setup 命令下的 Color 选项，并从上托选单中选 Menus，那么将出现如下选单：

```
Main/pull-downs
First pop-up
Second pop-up
Third pop-up
```

如果接着选 Main / pull–downs，那么可选择标题、边框、正文等的颜色。

最后，若从 Setup 命令下 Color 选项中选取 System boxes，那么下述内容将出现，

Index box
Directory box
Error box
Verify bos
Help
Status Line

其中条目要为下述元素设定颜色:

上托选单	将设定颜色的元素:
Input box	为文件名出现的提示(即，有关 File / Load或 File / Write to 选项的提示)
Directory box	在 File / Directory 选项中出现的目录盒
Error box	错误信息的打印
Verify box	为确认而作出的询问，即:
	Verify Overwrite TBCONFIG.TB? (Y/N)
Help	帮助屏幕
Status Line	Edit 盒中的第一行: 其中包含状态信息

上述 System box 中的各元素同样可用不同颜色显示。

选择要以一种颜色显示的元素时，一张颜色表出现在屏幕上，其中缺省颜色被框住. 此时，可先用光标键选取期望的前景色和背景色，然后再按 Enter 键.

象窗口的大小一样，颜色也可存入缺省配置文件 TBCONFIG.TB 或另一个扩展名为 .TB 的配置文件之中。

总之，通过用 Window 命令平分或堆积窗口，用 Scroll 键重新设定窗口大小和位置、以及用 Setup 命令下的 Color 选项设定颜色，用户可任意配置他（她）所喜欢的屏幕。

§ 9.8 Setup 命令下的 Miscellaneous(杂项)选项

§ 9.8.1 Auto Save Edit(自动保存编辑内容)

选取 Setup 命令下的 Miscellaneous Options 选项时，将出现下述带缺省设置的上托

选单:

Auto save edit	OFF
Backup source files	ON

如果设定 Auto Save Edit 项为 ON，那么 Turbo Basic 将在运行程序之前自动将它存盘. 这样将保护程序文件，使之避免在中断系统运行的运行错误发生时丢失. 若将 Auto Save Edit 设定为 ON，用户可通过将它存入文件 TBCONFIG.TB 或其他配置文件的方法，使这种修改永久化. 记住，Auto Save edit 选项只在选用 Run 命令时才保存文件；它不会在编译之后自动保存. 为了在编译之后自动保存当前文件，先按功能键保存、命名该文件，然后按 Esc 键脱离 Edit 命令. 另一种方法是，选取 File 命令下的 Save 选项.

§ 9.8.2 Backup Source Files(备份源文件)

源文件的备份是一个"第二手"或"备用"的程序拷贝. 假设用户创造一个叫做 Myfirst 的程序 (Turbo Basic 将它存作 Myfirst.BAS). 如果上托选单中的"Backup Source files"选项被设为 ON，那么计算机还将在一个名为 Myfirst.BAC 的备份文件中保存该文件.

假设用户将 Myfirst 装入内存，并在操作过程中由于动力事故或偶然将它删掉等原因使之丢失. 如果"Backup Source files"选项已设定为 ON (缺省设置为 ON)，那么，该程序的一个被称作 Myfirst.BAC 的备份，将在用户保存该程序之初创造. 然而，为了将备份文件用作 Turbo Basic 程序，必须在 DOS 状态下将它改名为: Myfirst.BAS.

为给该文件改名，在 DOS 状态下键入如下信息:

```
A> rename MyFirst.BAK MyFirst.BAS
↑
```

The prompt may be A, B, or C, depending on how your computer system is set up

注意，其中 DOS 提示符随计算机系统的设置不同，可能为 A、B 或 C 等.

根据当前所处驱动器的不同，文件名前面可能有 A:，B: 或 C: 等. 如果不想保存备份文件，那么将 Backup Source files 选项置成: OFF. 建议用户维护 Backup Source files 选项为 ON 的缺省设置，或将 Auto save edit 设为 ON. 这些都是减少程序丢失的简单保护方法. 如果用户在退出 Edit 方式之前总按 F2 键，那么 Auto Save edit 为 ON 的设置用途不大.

总之，Turbo Basic 不仅具有高效、用户友好的编译器，而且提供了许多为满足特定需要，适合各自"胃口"而配置屏幕和系统的选项.

§ 9.9 其他的 Turbo Basic 功能

§ 9.9.1 File 命令下的 Options 选项

§ 9.9.1.1 概述

在第二章中，我们只讨论了很少的几个 File 命令选项。下面为 File 命令下所有有效选项的摘要:

File 选项	意　义
Load	将一个事先保存的Turbo Basic程序装入内存。如果选用该选项计算机将提示用户输入要装入文件的名字。
New	清除Edit屏幕，以便新程序的输入。该选项将删除出现在Edit窗口中的任何程序，并为要输入的程序给定缺省文件名: NONAME.BAS。执行 New 选项时，所有四个窗口均将变为空白。如果在选用 New 选项时，所有四个窗口均将变为空白。如果在选用 New 选项之前正对另一个程序操作，并且未将它保存，那么 Turbo Basic 将询问用户是否想在当前工作程序从屏幕和内存中被清除之前将它保存。
Save	保存出现在Edit窗口中的程序。File命令下的Save选项可替代功能键 F2。后者用来在 Edit 模式下保存程序。记住，若以装入一个已存在文件作为处理过程的开始，则在对程序作过修改的过程结束时，应保存该文件。选用 Save 选项时，计算机用新的版本替换老版本。我们极力推荐，在脱离 Edit 模式之前，按功能键 F2 保存该程序。这样就不需要使用 File 命令下的 Save 选项。
Write to	为程序文件显式命名或改名，或者创造削前工各程序的附加拷贝。选用该选项时，Turbo Basic 将提示用户输入 New Name (新的文件名)。为创造一个新的程序文件，可先选用 New 选项，然后再选"Write to"选项，并键入一个程序名。或者，在处理过程刚开始时，先选"Write to"选项，然后为创建一个新的文件键入一个程序名。此外，假设当前工作程序为 Test1，那么还可通过选用"Write to"选项，并键入新文件名 Test2 的方法，创造该程序的"第二手"拷贝。
Main file	用于程序被分成各个程序块时。通常只供大型程序或其中包含存作各独立程序段（程序文件）的模块的程序使用。

Directory	显示存放在当前磁盘或Turbo Basic子目录中的所有程序文件。此时，用户应按提示"Enter mask（输入屏蔽字）"即输入程序文件的子集。例如，假设要列出以 test 开头的程序目录，那么键入 test＊.bas 作为对"Enter mask"的响应。Turbo Basic 将只显示其文件名以"test"开头的 Basic 程序名。"＊"被作"通配符"。如果在 Enter mask 提示下只简单地按 Enter 键，将显示当前目录（或子目录）中所有的文件。同样，键入"＊·＊"作为对"Enter mask"的响应时，将显示当前目录或子目录中存放的所有文件。注意，文件名按字母顺序列出。
Change dir	用于改变当前磁盘或子目录。例如，要想将一组程序存入B驱动器，那么先选取 Change dir 选项，然后在 New directory 提示下键入 B:。用这种方法，缺省驱动器便变为 B 驱动器。以后处理的任何文件都将假定为：供 B 驱动器使用。如果已改变了目录，那么，因为 B 驱动器为缺省驱动器，故程序或文件名前面不必带 B:。同样，要从 A 驱动器中装入一组程序时，用户可能希望将目录改为 A:，以便计算机自动从 A 驱动器中查找文件。记住，用户可以一直明确地使用文件名前缀指明要使用的驱动器或子目录。如果使用 Change dir 选项使 A 驱动器成为缺省驱动器，那么可将文件名写成：B: 文件名"，从 B 驱动器中读取文件。此外，也可以用该选项将程序存入不同的子目录。
Os shell	在内存中保留下Turbo Basic的同时，调入DOS操作系统。为了更改文件名、格式化磁盘、确定磁盘状态等，程序员常常选用该选项。执行完所要求的任何 DOS 操作后，键入 Exit 退回 Turbo BAsic。
Quit	彻底退出Turbo Basic，返回DOS状态下。如果未保存最后的工作程序，那么计算机将询问用户是否想将它保存。键入 Y 表示存，N 表示不存。Quit 选项所执行的功能与 Help 行上显示的 Alt-X 键所执行的一样；后者也将询问用户是否保存未存盘的文件。

§9.9.1.2 用 Load 选项从目录中选取程序

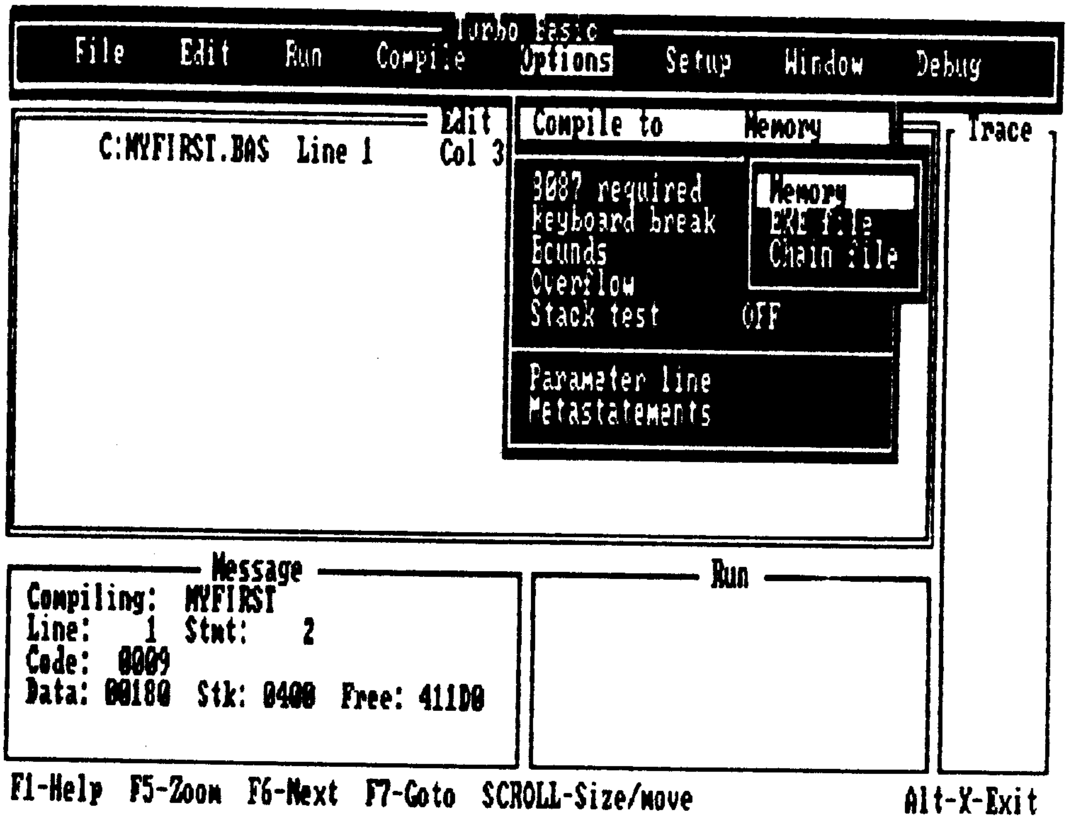
据我们所知，通过先选 File 命令下 Dirctory 选项，然后按 Enter 键响应"Enter mask"提示的方法，可显示一个目录。假设选用 File 命令下的 Load 选项，而且当时不记得要装入文件的具体名字，那么简单地按 Enter 键作为对要求输入文件名的响应，将得到缺省驱动器或子目录中存放的所有 Badic 文件按字母顺序排列的目录。

此外，也可以先用光标键加亮要装入的程序文件，然后简单地按 Enter 键选取该文件。总之，为了装入不记得其具体文件名的程序文件，键入 FL，并按 Enter 键。这样将得到所有 Turbo Basbic 文件的清单；此时，可用光标键加亮要装入的文件，然后按 Enter 键将它装入。

§9.9.2 进一步介绍 Options 选项

§9.9.2.1 保存程序的一个可执行的版本

假设已完全调试好一个程序，并要定期地运行该程序。回忆一下，Turbo Basic 将 Basic 源程序存作.BAS 文件。因为每次将它调回内存时都必须在执行之前编译，因而将程序存作.BAS 文件的效率低。而以可执行文件形式保存的效率要高得多。



为了以可执行文件形式保存文件，选取 Options 命令下的"Compile to"选项。此时，

可以看到其缺省设置为：Compile to Memory（编译到内存）；它表示程序从主内存中编译和运行。为了以磁盘上可执行文件形式保存编译后的程序，选取"Compile to"选项，并按 Enter 键（上述上托选单将出现）：

然后从"Compile to"上托选单中选取"EXE file（EXE 文件）"。编译后的程序将存入其文件名为原程序文件名，扩展名为.EXE 的文件中。即，如果调入程序 Myfirst，并将它编译成一个 EXE 文件，那么 Myfirst.EXE 将存放在缺省驱动器中。为运行该程序，完全不必装入 Turbo Basic 系统；只需在 DOS 状态下简单地键入 Myfirst，便可运行。如果已经在 Turbo Basic 环境中，则在主屏幕中（主选单中），键入 FO 回到操作系统状态下，然后键入 Myfirst 即可。

"Compile to"选项的上托选单也可以将程序或子程序存作随其他程序或子程序一起运行的链接文件（Chain file）第十二章讨论链接文件。

§9.9.2.2 其他选项

Options 选项还包括下列可打开（ON）或关闭（OFF）的选项：

选 项	意 义
8087 requried	在配有8087协处理器的计算机上运行程序时使用该选项。打开该选项（设为 ON）时，Turbo Basic 为浮点数操作生成最有效的代码。不过，请注意，该选项打开时生成的程序将不能在未配有 8087 协处理器的计算机上运行。
Keyboard Break	用来使程序员可按Ctrl-Break键中止程序的运行。关闭该选项（设为 OFF）时，如果程序有一个死循环，必须重新启动计算机才能退出该程序。如第二章所述，建议将该选项永久性地设为 ON。这就是说，在 TBCONFIG.TB 文件中存入 Keyboard break on"，改变缺省设置。
Bounds	在开辟表格或数组的程序中，Bounds（数组下标界限检查）应改为 ON。参见第十章。
Overflow	对于单精度整型变量，当算术运算产生超出-32768~+32767 的数值时，将导致数值溢出（overflow）。整型变量名的末尾均为%。如果程序在"Overflow ON（检查数值范围）"状态下编译，那么任何大于 32767 或小于-32768 的整型结果都将产生一个运行时错误:Overfolw（溢出）。例如，Overfolw 选项被设为 ON 时，下述程

序将导致运行时错误:

```
X% = 32000
PRINT X%+1000     '总和大于 32767
```

若在"Overfolw OFF"状态下编译上述程序，则一个错误的数值将赋给变量 X%，程序继续运行，而程序员或用户未得到发生错误的提示。建议将 Overfolw 设置成 ON，并将它存入 TBCONFIG.TB。

Stack	打开该选项（设为ON），使Turbo Basic生成检查码以确保有足够的内存空间供子程序使用。如果要编一个包含许多子程序、非常长的程序，并担心程序运行时内存不足，那么将该选项设为 ON。有一个称作 $ STACK 原语句的特殊语句可用来为子程序或其他程序分配更多的内存空间。

为改变上述选项中的任一选项，先将它加亮，然后，每按一次 Enter 键，该选项将由 OFF 变为 ON，再由 ON 变为 OFF，如此继续。如果打开一个选项，然后按 Esc 键，那么该设置一直生效，直至将它关闭为止。

其他两个选项，即，Parameter Line（命令参数行）和 Metastatements（原语句），通常不用于初级，甚至中级程序之中。

自我测验(二)

1. （T 或 F）Turbo Basic 屏幕底部的帮助行总是一样的
2. （T或F）因为F1总是具有相同的意义，即访问Help功能，因此它是一个热键。
3. 出现在主屏幕（Main Screen）帮助行上的 Scroll Lock 键用来____。
4. 编辑程序文件时，若按功能键____，文件将自动存盘。
5. 为永久性地改变 Turbo Basic 屏幕环境，必须将所作出的改变存入____文件。
6. （T 或 F）在屏幕上所有元素的颜色均可用 Window 命令下的 Color 选项改变。
7. （T或F）若从Setup命令下Miscellaneous选项中将Auto save edit选为ON，则程序将总是被自动保存。
8. 假设选用 File 命令下的 Load 选项，而不记得要装入程序的具体名字，怎么办?
9. （T或F）虽然File命令下的Save选项与功能键F2执行的操作相，但F2只有在Edit模式下才将程序存盘。
10. （T或F）程序一旦完全调试完毕，并计划定期使用，就应将它编译成其扩展名

为.EXE 的可执行文件，以便每次使用时不必编译。

§9.9.3 Edit 命令详述

§9.9.3.1 复习

在第二章中，已经介绍了输入程序和处理常见错误的基本正文编辑特性。

目 的	操 作
删除光标左边的一个字符 (例如，删除iq__中的q)	按BackSpace（回退）键（←） (例如，将显示q__)
删除光标处的字符 (例如，删除 iq 中的 q)	按Del（删除）键 (例如，将显示 i__)
删除光标所在的行	键入Ctrl−Y
将一行分成二行	用光标键将光标移至要显示在第二行的首字符，然后按 Enter 键（Ins 键必须打开）。 以下面语句为例: IF Amt<0 THEN Total=5 假设光标位于Total中的T处。如果当光标在Total中 T 处时按 Enter 键，那么将显示如下: IF Amt<0 THEN Total=5
缩进	按Tab键 (每按一次缩进8个字符)
插入一个字符 (例如，将 r 加入 pint)	先用光标键将光标移至插入处（如: Pint)，然后键入期望的字符（如: print)。在此，假设打开了插入状态，插入状态由出现在状态行上的单词"Insert"标注。
替换一个字符 (例如，将 psint 中的 s 改为 r)	用光标键将光标移至替换处（即，psint)；按 ctrl−V 或 Ins 键关闭插入状态。此时，屏幕上的单词"Insert"将被删掉。然后按期望的键（即，Print)。可为插入或替换式触发"Insert On")。

Turbo Basic 的编辑特性与 Wordstar 的非常类似.
本节将讨论有助于输入或修改程度文件的其他编辑特性.

§ 9.9.3.2 块操作

用户经常要对一系列指令或一个代码段（块）执行一组特定的操作. 例如, 重新安放程序块、将程序块复制到另一个段程序、删除程度块、保存程序块或打印程序块等, 为完成这些任务, 首先必须描述（定义）程序块, 然后再指出要执行的操作.

为了将一段程序定义为块, 必须在 Edit 方式下进行. 然后,

1. 用光标键将光标移到或突出块的起始处.
2. 接着按 F7 键.

   F7键标记块的起始处. 注意, 在Edit模式下, .help行表明F7用于指定"Beg Blk", 即块的起始处（beginning of block）.
3. 用光标键将光标移到或突出要定义为块的程序段末尾.
4. 接着按 F8 键.

   F8键标记块的末尾. 注意, 在Edit模式下, 屏幕底部的帮助行指出功能键F8的作用; 即, F8 用来指定"End Blk" —— 块的末尾（End of Block）.

   按F8时, 整个块将以反相形式或用不同颜色显著地标出. 具体形式取决于系统的配置的情况.

   定义好块以后, 可以将它移动、复制或进行其他操作. 假设要移动或拷贝该程序块. 移位时, 将块重新定位; 拷贝时, 在另一位置上复制该块.
5. 用光标键将光标移至或突出程序块要移到或拷贝的位置.
6. 按Ctrl-KV移动程序块, 或按Ctrl-KC拷贝程序块. 即, 按Ctrl键, 且在按住 Ctrl 键的同时按 K, 然后按 V 将块移动、或按 C 将块复制. 这些命令有时写作: ^KV 和^KC.

下面摘要给出块操作命令:

功　能	操作
拷贝块	Ctrl-CK（^KC）
移动块	Ctrl-KV（^KV）
删除块	Ctrl-KY（^KY）
从磁盘上读取块	Ctrl-KR（^KR） （此时, 提示输入要读入的文件名或块名）, 如果一个子程序要由一个单独的文件拷\入当前程序, 那么应使用该方法.

将块存入磁盘	Ctrl–KW (^KW) (此时，提示输入程序块要存入的文件名)。如果将一个要用于其他程序的子程序存盘，那么应使用该方法。
打印块	Ctrl–KP (^KP)
隐藏块	Ctrl–KH (^KH) (键入^KH可消除块中的高亮度。通常在选用其他块命令之后键入 Ctrl–KH，以便在它被操作完毕后删除对该块的任何引用。

§9.9.3.3 在程序中查找元素

在有的时候，可能要在程序中快速找出特定指令、变量名或标号。例如，假设要改变变量名。在这种情况下，用户希望能找出该变量名的每一个引用处，以便从头至尾、彻底地进行修改。同样，也希望不必翻遍整个程序便能快速找出程序的位置。如果在 Edit 模式下键入 Ctrl–OF，那么计算机提示用户输入要定位地词语或条目，然后提示输入任选项（将在后面介绍）。此时，如果按 Enter 键，那么计算机将找出指定条目的第一个引用处。再次键入 Ctrl—QF，将发现该条目的后续引用处。

因此，条目的搜索将从光标处开始。这就意味着，如果光标位于程序的最后一行，搜索将从那里开始，而不是从程序中的第一条指令开始。所以，如果要想找出对条目的第一个引用处，那么在进行条目搜索之前，应确信已将光标移至程序的开头。

于是，在编辑模式下将光标移至程序的第一行，然后按 Ctrl–QF 查寻一个条目。此时，计算机上出现提示"Find"；接着，输入要定位的词语或条目，按 Enter 键作为对查找任选项提示的响应，即不要任何选择；光标将指向指定词语或条目的第一个出现处。若接着按 Ctrl–L，则计算机将从光标处开始继续查找同一条目。结果，光标指向那个条目的下一个出现处。因此，如果使用后随 Ctrl–L 的 Ctrl–QF，那么可找出特定词语或条目的所有引用处。

§9.9.3.4 在程序中查找并替换元素

Ctrl–QA 键不仅能查找指定的条目，而且可对它们进行全局替换（global replacement。同样，在搜索之前（如果找出所有的条目），应确信光标位于文件的起始处。键入 Ctrl–QA 时，计算机提示用户输入要查找的条目："Find: "；按 Enter 键时，提示输入期望替换成的条目："Replace with: "；接着输入所要求的任何任选项（下节将详细讨论该任选项——在此，简单地按 Enter 键作为响应）。然后，Turbo Basic 将指向要求查找条目的各个出现处。并询问是否替换："Replace (Y/N)"。按 Y 进行替换。按^L 为后

续条目重复该查找和替换过程

查找命令小结

键	意　义
Ctrl-QF	找出条目对程序条目的第一个引用处
Ctrl-L	找出对程序条目的下一个引用处
Ctrl-QA	找出对程序条目的所有引用处，并进行全局替换．

如果指定的条目未找到，那么计算机将给出：

Search string not found. Press <ESC>

§9.9.3.5 查找和替换命令中的任选项

查找一个条目或查找并替换一个条目时，可如前所述按 Enter 键，不选取任选项．下面是用户可以选用的有效选择：

任选项	意义
B	从光标处开始往后（backward）搜寻．"Find"和"Find and Replace"的缺省设置为：从光标处往前（或下）搜寻．键入 B 响应"Find"所要求的 Options（任选项）时，搜寻将自光标处开始至文件起始处为止．
G	从文件起始处开始查找，并将光标定位于文件中所查条目的最后一个出现处．
U	忽视大小写字母的区别．如果指定："Find: HAL"并且 hal 出现在程序中，那么只有使用任选项 U 设定程序忽视大小写时，才可以找到 hal 或与 hal 匹配．
W	只找出整个词语与要查找条目匹配的情况．如果未指定任选项 W，要求查找 Amt，那第 Amt1，Amt2 等将作与 Amt 匹配的情况列出。为查找只与整个单词 Amt 匹配，而不与 Amt1 等匹配的情况，选用任选项 W．

如前所述，在查找或查找并替换过程中未找到匹配条目时，下述信息将出现：

Search string not found. Press <ESC>

§9.9.3.6 其他编辑命令：概述

下面是其他编辑命令的一个清单.

作 用	操 作
光标移动命令	
移至字符左边	Ctrl-S 或左光标键←
移至字符右边	Ctrl-D 或右光标键→
上移一行	Ctrl-E 或上光标键↑
下移一行	Ctrl-X 或下光标键↓
移至单词左边	Ctrl-A
移至单词右边	Ctrl-F
上卷一行	Ctrl-W
下卷一行	Ctrl-Z
上翻一屏	Ctrl-R或PgUp键
下翻一屏	Ctrl-C或PgDn键
扩展移动命令	
移至行首	Ctrl-QS或Home键
移至行尾	Ctrl-QD或End键
移至窗口顶部	Ctrl-QE
移至窗口底部	Ctrl-QX
移至文件起始处	Ctrl-QR
移至文件末尾	Ctrl-QC
移至块首	Ctrl-QB
移至块尾	Ctrl-QK
移至最后的光标处	Ctrl-QP
插入和删除命令	
开／关插入模式	Ctrl-V或Ins键
插入一行	Ctrl-N
删除一行	Ctrl-Y
删到行尾	Ctrl-QY
删除光标左边字符	Ctrl-H或Backspace键

删除光标处的字符	Ctrl-G或Del键
删除单词的右边	Ctrl-T

其他命令	
退出编辑；但不存盘	Ctrl-KD，Ctrl-KW，或Esc
保存并编辑	Ctrl-KS，或F2键
读入新文件	F3键
制表	Ctrl-I，或Tab键
制表模式	Ctrl-OT
开／关自动缩进功能	Ctrl-OI
恢复一行	Ctrl-QL
设置位置标记	Ctrl-KN
查找位置标记	Ctrl-QN
查找	Ctrl-QF
查找并替换	Ctrl-QA
重复上次查找	Ctrl-L
控制字符前缀	Ctrl-P
放弃操作	Ctrl-U
恢复错误信息	Ctrl-QW

§9.10 本章辅导

§9.10.1 本章小结

Ⅰ. Turbo Basic 环境

A. Help 功能

1. 在 Turbo Basic 中按功能键 F1 总可得到帮助
2. 帮助功能是与上下文有关的，即帮助信息与当前工作的特定命令有关。
3. 在任何时候，按两次F1键可得到用户可从中选取条目的帮助选项索引。

B. 命令和操作

1. 命令和选项的选择。
   a. 先用光标键加亮期望的特性项，然后按 Enter。
   b. 或者，先键入命令首字母，接着输入选项的首字母。
2. 选单
   第一级选项在下拉选单中显示；所有低级选项在上托选单中显示。

C. 运行状态

当前屏幕或当前窗口具有双线边框。所有的其他窗口均为单线框窗口。

Ⅱ. 重新配置屏幕.

A. Tile（平分）

为使所有的四个窗口大小相等，选用 Window 命令下的 Tile 选项.

B. Stack（堆积）

为使窗口相互重叠，一个窗口位于另一个窗口上面，选用Window命令下的Stack 选项.

C. 通过先使窗口成为当前窗口，然后选用Window命令下的Close选项的方法，可将窗口从屏幕显示中清除. 采用这样方法，可以从屏幕上清除任意数目的窗口. 为重新显示已关闭的窗口，仅需简单地将它 Open（打开）即可.

Ⅲ. 重新设置窗口大小和位置

A. 在主屏幕上按 Scroll Lock 键开始重新配置.

B. 然后一起按 Shift 和光标键重新设置当前窗口的大小.

C. 不按Shift键，仅按光标键重新确定当前窗口的位置，以使它出现在屏幕上不同的地方.

D.再次按 Scroll Lock 键关闭重新配置窗口大小和位置的功能.

Ⅳ. 在屏幕上显示颜色

用 Setup 命令下的 Color 选项修改屏幕上的颜色配置.

Ⅴ. 保存配置文件

A. TBCONFZG.TB为保存Turbo Basic系统配置或环境的缺省配置文件. 装入Turbo Basic 系统时，该文件指定屏幕的显示形式.

B. 如果对Turbo Basic工作环境作了改变，必须将它存入TBCONFIG.TB文件，以便使之永久化.

C. 也可以创造替换的配置文件；它可为特定应用场合装入使用.

Ⅵ. 其他命令复习

A. File

1. Load——装入一个已存在的程序文件.
2. New——清除屏幕，并输入新程序.
3. Save——保存程序；与 Edit 方式下的功能键 F2 功能相同.
4. Write to——将文件存入新的文件名.
5. Main file——将若干程序链接在一起.
6. Directory——显示文件名清单.
7. Change dir——用不同的磁盘或子目录装入或存写文件.
8. Os Shell——临时脱离TB环境，执行一些DOS操作. 为返回Turbo Basic

键入 EXIT.

9. Quit——彻底退出 TB 环境.

B. Options

为程序每次运行时不必重新编译，从Options命令中将"Compite to"选项设为 EXE，即"Compile to EXE file"，把程序编译成.EXE 文件.

C. 编辑

一系列WordStar文本编辑特性可用于编辑、块操作、查找和替换、光标控制等方面.

§ 9.10.2 本章自测

1. (T 或 F) 在 Trace 窗口中可进行放缩操作.
2. (T 或 F) 一个已关闭的窗口在未重新启动系统时不可能被重新打开.
3. (T或F) 用Setup命令将选项存入TBCONFIG.TB文件，可永久性地修改窗口大小和位置.
4. 如何改变当前窗口的大小?
5. 如何改变当前窗口的位置?
6. 如何平分所有的四个窗口?
7. 如何从屏幕上删除一个窗口?
8. 指出扩展名.EXE 的意义?
9. 指出扩展名.TB 的意义?
10. 指出扩展名. * 的意义?
11. 指出扩展名.BAK 的意义?
12. 如何只平分 Run，Edit 和 Trace 窗口?
13. 如何只堆积 Run 和 Edit 窗口?
14. 如何在 Edit 方式下保存文件?
15. 如何结束 Turbo Basic 处理过程?

§ 9.10.3 复习题

若已装入 Turbo Basic 系统，用"笔试"的方法回答下述问题:

1. 哪些命令有选项，哪些没有?
2. 在主屏幕上键入FO，将出现什么? 然后键入EXIT，又将发生什么? EXIT的目的何在?
3. Alt-X 与____产生同样的结果.
4. 在 Edit 模式下按功能键 F2 产生与____相同的结果.
5. 按两次 F1 键显示什么? 如何选取你想选取的选项?
6. 键入三行代码，在Edit模式下，将光标指向第二行行首；键入^KB，再将光标移

至该行行尾并键入^KK，接着将光标移至第四行，键入^KV．请问，你完成的任务是什么？说明如何用功能键完成同样的操作．

描述在 Edit 模式下下述各键（或组合键）的作用：

7. F1
8. F2
9. F6
10. F7
11. ^KS
12. ^QG
13. Home
14. End
15. PgUp
16. Alt-X
17. Alt-R
18. PgDn
19. F8
20. F3

§9.10.4 编程题

1. 打印磁盘上所有 Turbo Basic 程序的名字．
2. 拿一个Turbo Basic程序在磁盘上创建一个可执行程序或目标程序，并运行该程序．
3. 用 Turbo Basic 创造一个程序的备份．
4. 装入一个Turbo Basic程序，将它运行，并打印该程序的平分屏幕和Run窗口．
5. 重新配置屏幕，使Run和Edit各用屏幕的一半．即清除Trace和Message窗口，并重新设置 Edit 和 Run 窗口的大小，然后装入一个 Turbo Basic 程序，将它运行，并打印重新配置后的屏幕．
6. 将上面的重新配置屏幕存入一个可替换的配置文件．重新启动Turbo Basic系统并重新装入这个新的配置文件；然后装入另一个 Turbo Basic 程序，并将它运行，打印屏幕．
7. 装入一个Turbo Basic程序；打印它的放大版本(放大后的Edit窗口)．然后运行该程序，并打 RUN 窗口的放大形式．
8. 打印一个空的平分屏幕，其中 Message 窗口为当前窗口．
9. 若有一个彩色监视器，重新配置平分屏幕，以使各窗口用不同颜色显示．
10. 进入Help状态下，并从Help索引中选取"Compile Command"帮助屏幕．打印与"Compile Command"的帮助屏幕．在这个屏幕底部标有"Go Dn … "是什么意思？如何处理？如何退出 Help 状态？
11. 装入一个 Turbo Basic 程序

a. 将前三行移至程序末尾.

b. 将最后三行复制到程序起始处.

c. 删除最后三行. 此时，结果应为开始处理时的程序.

12. 装入一个在屏幕上显示输出的Turbo Basic程序，用查找并替换命令将其中的PRINT 指令全部换成 LPRINT.

13. 计划编写一个程序，以在主选单中键入F作为开始. 此时，得到一个下拉选单，但试图用光标键加亮 Write to 选项时，未能正常工作. 此时 NUM 出现在帮助行上. 找出问题所在并改正.

14. 编写一个程序，提示分三行输入每个顾客的姓名和地址；在打印机上为各顾客打印五个邮递标签. 编程时，不用子程序，以便在程序起始处和 DO WHILE … LOOP 结束处均为打印和输入语句. 然后，用块操作命令将这些输入指令中的一组放入子程序中，然后删除第二组，并换成正确的 GOSUB 语句:

```
.
.
.
GOSUB InputRoutine
.
.
.
DO  WHILE
    .
    .
    .
    GOSUB InputRoutine
LOOP
END
InputRoutine:
    .
    .
    .
RETURN
```

第 五 部 分
Turbo Basic 的高级特性

本 部 分 内 容

第 十 章　数组及表格处理
第十一章　用过程进行自顶向下模块化程序设计详述
第十二章　自顶向下模块化程序设计中的
　　　　　函数、字符串处理及递归
第十三章　文件处理和数据库管理
第十四章　图形和声音

第十章　数组及表格处理

本章要求:

1. 熟悉如何建立数组.
2. 熟悉如何将数据读入数组及如何处理.
3. 熟悉如何在数组中存储总和.
4. 熟悉如何利用数组进行表格处理.

1A8-32(2)

§ 10.1　数组介绍

在程序设计中使用数组来表示重复出现的变量，其数据元素可以全部是数字式的，也可全是字符串式的变量. 数组最广泛的用途在于:

> **数组的用途**
>
> 1. 存储一组具有相同类型的变量（例如: 全是字符串变量或全是数字变量）.
> 2. 存储一组输入数据的总和; 在全部输入数据累加后，可打印输出总和.
> 3. 存储用于"检索"的表格.

本章将讨论数组在这三方面的用途.

§ 10.2　数组用于存储数字或字符串变量

例: 假如要读入并处理一个班20名学生的考试成绩; 每个学生一个成绩. 若使用传统方法，就得分别输入 20 个变量，每个成绩要用一个变量.

```
INPUT Exam1
INPUT Exam2
:
:
INPUT Exam20
```

此外，若要求这 20 个成绩的平均值，则还得编一串冗长的代码:

```
Average= (Exam1+Exam2 ··· +Exam20) / 20
```

上述语句中的省略号（…）意即：程序员需对20个独立的输入数据重复编码．这20个考试成绩都是数字量，可用单一名字的变量来定义，看作是一个数组变量．

§10.2.1 DIM语句定义数组的大小

在数组中存放一组数据元素，可用DIM语句定义．要在一个数组中存储20个考试成绩，可编程如下：

```
DIM  Exam (1：20)
```

下面是如何用含20个元素的Exam数组存储数据的例子：

DIM Exam (1:20)

1st Exam	2nd Exam	...	20th Exam
Exam (1)	Exam (2)	...	Exam (20)

Exam称作数组(array)；各个变量Exam (1)，Exam (2)，…Exam (20)称作数组的元素．在同一数组中所有元素必须是同一类型的；本例中，各元素都是数字变量．DIM Exam (1：20)中，1表示第一个元素(1)，20表示最后一个元素(20)．

DIM语句

数组定义语句DIM应该放在所有使用数组语句的前面，下面的次序是不正确的．

错误：DIM必须在赋值语句的前面．

```
Let Exam (20) = 0
    :
    .
DIM Exam (1:20)    'DIM must precede the assignment above
```

注意：并非所有数组都要用DIM语句，如果你在程序中未用DIM语句但使用了数组，Twrbo Basic将默认最大为11个元素，同时又对元素编号为Exam (0) - Exam (10)，这可能会引起混淆．

```
INPUT Exam (0)        'This is the first element of the array
INPUT Exam (1)        'This is the second element of the array
   :                  '11 is the maximum array
   .                  'size unless you use a DIM
INPUT Exam (10)
```

如果在上面的INPUT指令前未加DIM语句，即缺省DIM Exam (10)，则元素编号Exam (0) … Exam (10)．在使用Exam (I)时，只要不使I>10，就不会出错误

信息；即：对代码 INPUT Exam（11），如前面没有 DIM 语句，就会引起语法错误.

DIM Exam（19）与 DIM Exam（1∶20）都定义了 20 个元素的数组，但 DIM（19）其元素是 Exam（0） ··· Exam（19），而 DIM Exam（1∶20）其元素是 Exam（1） ··· Exam（20）；即 DIM Exam（19）定义了从 0～19 的 20 个元素的数组，DIM Exam（19）圆括号中的数比其定义的变量数少 1；这一怪现象是因为 DIM variable（n）语句定义了从 0～n 编号的 n+1 个变量.

大多数程序中，都用 DIM Variable（n∶m）；这里 n 表示数组的下限，m 表示数组的上限；因此，DIM Exam（1∶20）建立 20 个元素的数组．确切地讲，是从（1）～（20）变化.

编程指导（CODING GUIDELING）

为确保处理得当，编程清楚，常使用 DIM 语句；其格式为：DIM array（n∶m）．n 表示数组元素的最小编号值，m 表示数组元素的最大编号值.

§10.2.2 使用下标访问数组元素

下面分析用 DIM Exam（1∶20）定义的包含 20 个元素的数组 Exam；假定已输入数据给数组，要访问这 20 个数组元素中的任一元素，应该用数组名 Exam 以及带有元括号的数字；该数字称作下标（subscript）．下标分别指向要访问的 20 个输入元素（1–20）．例如，我们用 Exam（1）访问第一个学生的成绩，用 Exam（2）访问第二个学生的成绩，以此类推．如果这 20 个考试成绩已经录入并储存到计算机中，则可通过如下编程打印第五个学生的成绩：

```
PRINT "GRADE FOR STUDENT 5 IS"; Exam (5)
```

下标指定了要访问的数组元素；在上面的 DIM Exam（1∶20）中，下标是要访问的实际元素编号．因为下标通常从 1–m 变化，m 是数组元素的编号.

数组与下标结合使用：

数组与下标

1. 用DIM语句定义数组类型：DIM ArrayName（n∶m），数组表示同一类型的数据重复出现的变量.
2. 假如编程为：DIM ArrayName（n）．程序要指出所访问数组元素的某个数据项，下标应该从 0–n；ArrayName（0）表示第一个元素，ArrayName（1）表示第二个元素， ··· ArrayName（n）表示第 n+1 个元素．若没有 DIM 语句，编译时则假定仅有 11 个数组元素，下标从 0–10.

数组名和下标结合起来，才能访问数组中的任一元素；例如程序中定义了 DIM Exam (1：20)，则数组 Exam 的下标值为 1–20，其它值就会导致错误:

无效语句:

```
PRINT Exam (0)
PRINT Exam (21)
```

因为所定义的数组中没有第 0 号，第 21 号元素，下标不能取 0、20. 此例中，下标只能取 1–20 的整数. 一般地，下标只取由 DIM 语句规定范围的整数；如果是带有十进制小数（例 3.2），将把它化成整数. 注意，对 n.5 化整时，如果 n 是奇数，则向上进位；如果 n 是偶数，则不进位. 故 Exam (3.2) 指的是 Exam (3)，Exam (3.5) 和 Exam (4.5) 则都指 Exam (4).

§ 10.2.3 Option Base 语句

如果程序中有 DIM 语句，或 DIM 语句为 DIM ArrayName (n) 的形式，则数组中第一个元素的下标为 0 值. 然而，正如我们已看到的那样，数组 Exam 的第一个元素为 Exam (0)，第二个元素为 Exam (1)，等等；这可能容易引起混淆.

因此把 1 作下标来访指第一个元素，2 作下标访问第二个元素等等，就显得比较清楚. 为此，可使用 DIM (n：m) 语句. 或者，也可通过在 DIM 语句前编制如下程序方法，把所有数组起始下标值均改变为 1，而不是 0.

```
OPTION    BASE    1
DIM    Exam (20)
```

OPTION BASE 指示所有数组的第一个元素之下标值. 使用上面的 OPTION BASE 语句后，所定义的 20 个 Exam 数组元素就成为 Exam (1) –Exam (20)；DIM (20) 就只有 20 个元素了. 下面的几种情况是等价的.

建立一个 20 个元素数组的方法

```
1. DIM Exam (1:20)
2. OPTION BASE 1
   DIM Exam (20)
3. DIM Exam (19)          'Least preferred method
```

用方法 1 和 2，每个元素由指向数组逻辑位置的下标确定:

1st Exam	2nd Exam	...	20th Exam
Exam (1)	Exam (2)	...	Exam (20)

因为它能使程序设计容易、可读性好，所以要用 OPTION BASE 1 或 DIM (1∶20) 语句；OPTION BASE 语句的上限为 32767. 注意，OPTION BASE 1 语句针对程序中所有定义的数组.

§10.2.4 下标可以是整数、变量或算术表达式

到目前为止，我们集中讨论了下标是整数的情况；但是，下标也可以是整型变量. 假设定义如下的 Exam 数组和数字变量.

```
Option Base 1
DIM Exam (20)
LET I = 1
```

则可用下面任一方法打印第一个考试成绩单:

```
PRINT Exam (I)
```

```
PRINT Exam (1)
```

就是说，数组中表示特定项的下标既可以是整数，也可是整型变量. 事实上，最好是用变量名 I%来指示下标变量；I%变量总是取整数值. 如果 I%误输入为 3.4，它将取 3. 不过，要注意的是，在一个程序中既用 I 又用 I%，它们将被看作是两个不同的变量.

使用变量而非固定的整数作下标，则可改变下标；如此可在单一过程内处理一串数据. 假定要求 20 个成绩的平均值，用整数作下标，一点也省不了编程代码.

Average= (Eaam (1) +Exam (2) + ··· +Exam (20) / 20

这远不如用一个循环一次累加出总和效率高；其中用到下标变量，可以由 1～20 改变下标值，这样 20 个考试成绩就累加起来了.

§10.2.5 用 FOR ··· NEXT 处理数组

假如所有 20 个考试成绩都已存储在 Exam 数组中，则求平均值的过程可用

FOR–NEXT 循环编程如下:

```
'Calculate an Average assuming Exam1-Exam20
'Have already been entered
CLS
DIM Exam (1:20)
Total = 0
FOR I% = 1 TO 20
    Total = Total + Exam (I%)
NEXT I%
Average = Total / 20
PRINT "Average = "; Average
END
```

用 FOR–NEXT 给下标一个初值和增量，直至整个数组处理完毕；也可用其它类型的循环，但它们还得用其他指令来给下标变量赋初值和增量，而 FOR–NEXT 循环则可自动完成这一工作。

> **编程指南**
>
> **诸如 FOR … NEXT，DO WHILE，WHILE … WEND，或 DO UNTIL 之类的任一迭代结构都可用于处理数组元素。在处理全部的数组元素时，建议使用 FOR … NEXT 语句。这种下标变量能自动赋初值，且可以给出相应的增量；如此可使编程清晰，而又简短。**

我们再来考虑先前的两个数组处理的例子:

例 1 找出最好成绩?

要找出该班最好的成绩，可编制如下程序:

```
'Find highest grade
'Assume array already entered
CLS
DIM Exam (1:20)
Highest = 0              'Initialize Highest
FOR I% = 1 TO 20
    IF  Exam (I%) > Highest THEN
        Highest = Exam (I%)
    END IF
NEXT I%
PRINT "Highest Grade is "; Highest
```

例 2 找出成绩最高的学生是谁?

要打印出分数最高的学生序号，需要保存最好成绩及其下标。

```
'Find Student with the highest grade
'Assume array already entered
```

```
CLS
DIM Exam (1:20)
Highest = 0
FOR I% = 1 TO 20
    IF  Exam  (I%) > Highest THEN
        Highest = Exam (I%)
        Student = I%
    END IF

NEXT I%
PRINT "Student "; Student; " Has Highest Grade "
```

例 3 找出大于等于 80 分学生的数目:

```
'Find the number of students
'with a grade of 80 or more
'Assume array already entered
CLS
DIM Exam (1:20)
Count = 0
FOR I% = 1 TO 20
    IF  Exam (I%) >= 80 THEN
        Count = Count + 1
    END IF
NEXT I%
PRINT "Number of Students with a Grade of 80 or More is "; Count
```

§ 10.2.6 把数据读入数组

到目前为止，我们一直假定数据已读入 Exam 数组；现在，我们来看录入 20 个考试成绩的过程。

```
'Input data to an array
DIM Exam (1:20)
FOR I% = 1 TO 20
    PRINT "Enter Exam Score "; I%
    INPUT Exam (I%)
NEXT I%
```

注意：数组不一定要初始化为 0 值，因为要将输入数据赋给它。

例 1.把 7 个月销售总额读入数组。

```
OPTION BASE 1
DIM Sales (7)
FOR I% = 1 TO 7
    PRINT "Enter Sales Total for Day "; I%
    INPUT Sales (I%)
NEXT I%
```

例 2.假定上例数组已赋值：计算周销售总额.

```
Total = 0
FOR I% = 1 TO 7
    Total = Total + Sales (I%)
NEXT I%
PRINT "The Total Sales is "; Total
```

例 3.假定题 1 中数组已赋值，打印日销售总额超过 $ 1000.00 的总天数.

```
Ctr = 0
FOR I% = 1 TO 7
    IF  Sales (I%) > 1000 THEN
        Ctr = Ctr + 1
    END IF
NEXT I%
PRINT "Total No. of Days with Sales > 1000 is "; Ctr
```

自我测验(一)

考虑下面几个语句：

```
INPUT Var1
INPUT Var2
INPUT Var3
INPUT Var4
INPUT Var5
```

1. 因为______，数组可用来取代各个Var.
2. 假定Var3和Var5可定义成Var3 $ 和Var5 $；也即，它们是字符串变量，能用一个数组来定义这 5 个变量吗？说明理由.
3. 对上面指令重新编码，用数组来代替变量，并输入这5个数值.
4. 假定要建立一天24小时温度的数组，可定义一数组DIM Temp（23）：它允许接收从____到____的元素.
5. 上面元括号中的数字称作____.
6. 为使元素从____到____，使用OPTION BASE 1语句和DIM（1：24）更容易些.

假定下面的数组已储存数据：

```
OPTION  BASE 1
DIM  Temp (24)
```

7. 写出求该日温度均值的程序.
8. 写出求该日最低温度的程序.
9. 写出打印温度最低点时间的程序.
10. 编程求一天中温度低于20℃的小时数.

§ 10.3 数组和下标的几点补充

§ 10.3.1 数组中的元素不一定要连续处理

下标可能是算术表达式，因此，在 Exam 数组中要求偶数编号成绩的均值，可编程如下.

```
Total = 0
DIM Exam (1:20)
FOR X% = 1 TO 10
    Total = Total + Exam (2*X%)
NEXT X%
Average = Total / 10
PRINT "The Average For Even Numbered Exams Is "; Average
END
```

算术表达式 2 * x%用作下标，称作相对下标（relative Subscripting).

同样，也可倒序处理数组元素；假定要输出 Exam20, Exam19 … Exam1，可编程如下:

```
'Assume array is already stored
Dim Exam (1:20)
FOR I% = 20 TO 1 STEP -1
    PRINT "Exam "; I%; "Is "; Exam.(I%)
NEXT I%
```

§ 10.3.2 数组中元素能定义一个范围,可以不是 1～n

不同于其它大多数 Basic 版本，数组元素只能定义为固定的序号从 1（或 0）到 n. Turbo Basic 允许给元素定义一个范围，它可以不正好到上限值；例如，若希望存储 1981–1990 年的人口数据，可建立可下数组:

```
OPTION  BASE 1
DIM  PopulationData (10)
```

这样，就得记住，PopulationData（1）是指 1981 年而 PopulationData（2）是指 1982 年，等等；最好的办法是按下面编程:

DIM PopulatinData (1981 : 1990)

这样便产生一个编号为 1981~ 1990 的 10 个元素的 populationData 数组 populationData (1981) 清楚地标明为 1981 年人口数据, populationData (1982) 指 1982 年人口数据, 等等.

要输入人口数据，并计算十年平均人口，可编程如下:

```
'Enter population data for 1981-1990
'Calculate Average
CLS
Total = 0
DIM PopulationData (1981:1990)
FOR Year% = 1981 TO 1990
    PRINT "Enter Population For "; Year%
    INPUT PopulationData (Year%)
    LET Total = Total + PopulationData (Year%)
NEXT Year%
Average = Total / 10
PRINT "Average Population is "; Average
```

§ 10.3.3 由字符串变量组成的数组

注意: 数组既可用来存储字符串变量,也可用于存储数字变量; 可用 OPTION BASE 1 定义如下

```
OPTION  BASE  1
DIM  CustName$ (10)
```

或按如下编程:

```
DIM  CUstName$ (1 : 10)
```

自我测验(二)

先看下列语句:

```
OPTION  BASE  1
DIM  Exam(20)
```

1. 编程求成绩大于或等于 90 分的学生总数.
2. 该班成绩存储在Exam数组中，假定该班成员希望以80分为界，画成绩曲线; 如平均成绩大于 80，则不画曲线. 若有的话，编程确定该曲线的形式.
3. 使用数组编一段程序，读入LosAngles某一天24小时的气温.

4. 假定上面数组的第一个输入为1.Am，第2个输入为2.am，依次类推；编程求一天中达到最高气温的时间。
5. 求下午1：00到午夜0:00的平均气温。
6. 把50个州人口数据读入一个数组，每个州一个数据，计算美国总人口。
7. 把1992年DowJones每日平均工业产值读入数组，计算并打印DowJones平均工业产值低于 7800 的天数；注:1992 年有 366 天。

§ 10.4 使用数组存储总和

§ 10.4.1 概述

迄今为止，我们已了解了数组既可用于存储输入变量，也可用于存储输入变量的累加和；在全部数据输入后，把数组打印出来。

例如：假定一个公司有 25 个推销员，一个推销员每次只做成一个交易；输入推销员的编号 1-25 和销售量。对有些推销员可能会有大批量的输入值，在把所有 25 个推销员的销售数据全部录入后，希望打印出每人一周的总销售量。每一个输入对应一次交易，交易可随时输入，不按顺序输入，而是间断的录入；第一个输入项可以是 03 号推销员，紧接着为 22 号，等等。如果从 01、02 号记录依次把所有的推销员数据全部录入，工作可能会容易些。这样，就可统计 01 号推销员的销售量,并打印出结果,然后统计 02 号，等等。正如第七章所述，该过程可能是一个中断控制过程。

因为录入不按顺序存放，用户必须建立存放 25 个总和的数组。

```
DIM  Sales  Total  (1 : 25)
```

该语句定义了 25 个元素的数组 Sales Total，用来存放 1-25 号推销员销售量的累计和。第一行输入推销员编号，第二行输入该推销员的销售量；然后，将这些销售量累加放到数组 SalesTotal 相应的元素中，由已录入的销售人员的编号确定存放累加和数组的相应元素。

§ 10.4.2 储存总和的数组必须初始化

在累加销售量之前，应该对数组 SalesTotal 进行初始化，下面的程序用来把 SalesTotal 数组初始化为 0，用 I%作下标。

```
DIM   lesTotal (1:25)
FOR I% = 1 TO 25
    SalesTotal (I%) = 0
NEXT I%
```

初始化数组中全部元素最简单的办法是用 ERASE 语句:

ERASE arrayname〔, arrayname〕 …

ERASE 语句将把数字量数组元素全部设置为 0，把字符串数组元素都置成空串.

§ 10.4.3 把输入数据累加到数组

下面是累加推销员销售总量的伪代码:

```
START
        初始化数组 SalesTotal
        输入售货员号
        输入交易量
        DO  WHILE  还有数据?
            将交易量加入相应的数组元素
            输入售货员号
            输入交易量
        END  DO
        打印标题
        打印数组内容
STOP
```

程序员可使用 SalesNoIN 下标，决定 SalesTotal 的哪一个元素存放输入销售量的累计值；这个下标也可以是销售人员的编号字段. 只要是 1–25 之内的有效值，那么该下标就可作为输入变量.

假定第一个输入变量输入如下:

```
ENTER SALESPERSON NO:
03
ENTER TRANSACTION AMOUNT:
123.50
```

因为下标号 SalesNoIn 是 03，故可把累加值相应地放入 Sales Total 数组的第三个元素中；程序设计如下:

```
DIM SalesTotal (1:25)
PRINT "ENTER SALESPERSON NO:"
INPUT SalesNoIn%
PRINT "ENTER TRANSACTION AMOUNT:"
INPUT TransAmt
SalesTotal (SalesNoIn%) = SalesTotal (SalesNoIn%) + TransAmt
```

§ 10.4.4 累加到数组前要有效化输入数据

用作下标的输入变量 SalesNoIn%，必须在 1-25 之内变化，如果 SalesNoIn%小于 01 或大于 25，就会出现错误．设计程序应包括有效化的检查，以免处理过程中发生错误．

```
IF  (SalesNoIn% > 0) and (SalesNoIn% < 26) THEN
    SalesTotal (SalesNoIn%) = SalesTotal (SalesNoIn%) + TransAmt
ELSE
    PRINT "Input Is In Error: Reenter"
END IF
```

SalesNoIn%是一个变量，录入后，将用作整型下标。上面的程序使 SalesNoIn%有效化，以确保其取值在 1-25 之间；若不然，则打出错误信息．如果省掉这一确认程序段，则当 SalesNoIn%中录入不正确值时，运行中将会出现错误．

§ 10.4.5 打印数组数据

编制一段读入 SalesNoIn%和 TransAmt 的程序，且当 SalesNoIn%有效时，该程序把 TransAmt 累加到 SalesTotal 数组的相应元素中．所有数据处理完后，打印出 25 个销售总量，销售总量连续地从 01-25 依次打出．打印程序用 FOR … NEXT 语句，下标用 I%，从 1-25 变化．该循环打印类似初始化数组，依次遍访整个数组。这样，程序中就有两个下标：一个是 SalesNoIn%用来指定累加值存入数组中哪个元素；另一个是 I%，用来给数组清零，并以循环方式按顺序打印 25 个累计值．

例 1.下面是整个程序:

```
'Program enters Transaction Amounts for
'Salespersons 1-25 in any sequence
'And prints the total for each salesperson
'in sequence
CLS
DIM SalesTotal (1:25)
FOR I% = 1 TO 25
    SalesTotal (I%) = 0          'Initialize SalesTotal
NEXT I%
MoreInput$ = "Y"
DO  WHILE MoreInput$ = "Y"
    GOSUB PromptAndGetInput
    IF  SalesNoIn% > 0 AND SalesNoIn% < 26 THEN
        SalesTotal (SalesNoIn%) = SalesTotal (SalesNoIn%) + TransAmt
    ELSE
        PRINT "Input is in Error: Reenter"
    END IF
    PRINT "Is There More Input (Y/N)?"
    INPUT MoreInput$
```

```
LOOP
FOR I% = 1 TO 25
    PRINT "Total for Salesperson "; I%; "Is "; SalesTotal (I%)
NEXT I%
END
PromptAndGetInput:
    PRINT "ENTER SALESPERSON NO:"
    INPUT SalesNoIn%
    PRINT "ENTER TRANSACTION AMOUNT:"
    INPUT TransAmt
RETURN
```

例 2.假定有如下格式的工资单记录文件:

EmpNo, EmpName$, AnnSal

公司经理要一份雇员数目的百分比的报告.

工资范围	雇员总数	雇员百分比数 (%)
< $25000		
$25000–40000		
$40000–50000		
> $5500		

建立一个四元素的数组,把工资在指定范围内总的雇员数存起来.

< 25000	
25000-40000	
40000-55000	
> 55000	

建立该数组并把数组初始化为0值后，再读入雇员记录；根据每个雇员所挣工资的情况，把它累加起来存入相应的数组元素．待所有的雇员记录输入完毕后，打印出雇员总数及每个工资段雇员所占的百分比．图 10.1 是包含输入及输入出样例的程序．

注：程序开始，应该对所有数组定义．

```
'****************************************
' This program prints the total number
' of employees and the % of employees
' who fall into four earnings categories
'****************************************
DIM SalArray(1 TO 4)
DIM Range$(1 TO 4)
DIM Percents(1 TO 4)
GOSUB Initialize
GOSUB ReadModule
```

```
DO WHILE MoreRecords$ = "Y"
        Total = Total + 1
        IF Salary < 25000 THEN
                SalArray(1) = SalArray(1) + 1
        ELSEIF Salary < 40000 THEN
                SalArray(2) = SalArray(2) + 1
        ELSEIF Salary < 55000 THEN
                SalArray(3) = SalArray(3) + 1
        ELSE
                SalArray(4) = SalArray(4) + 1
        END IF
        GOSUB ReadModule

LOOP
GOSUB PrintResults
CLOSE #1

END
'****************************************
' This subroutine defines the array,
' initializes it and other variables
'****************************************
Initialize:
        CLS
        FOR I% = 1 TO 4
                SalArray(I%) = 0
        NEXT I%
        MoreRecords$ = "Y"
        Total = 0
        OPEN "Employee.dat" FOR INPUT AS #1
RETURN
'**************************************
' This subroutine reads data and tests
' for the end of the file
'**************************************
ReadModule:
        IF EOF(1) THEN
                MoreRecords$ = "N"
        ELSE
                INPUT #1, EmpNo, EmpName$, Salary
        END IF
RETURN
'**********************************************
' This subroutine prints the results - note
' that is uses two additional arrays that
' the four ranges that are to print and the
' four percentage
'**********************************************
PrintResults:
        LPRINT CHR$(12)
        LPRINT TAB(40); "SALARY REPORT"
        LPRINT
        LPRINT
        LPRINT TAB(10); "SALARY RANGE";
        LPRINT TAB(30); "TOTAL NO. OF EMPLOYEES";
        LPRINT TAB(60); "% OF EMPLOYEES"
        LPRINT
        LPRINT
        FOR I% = 1 TO 4
                READ Range$(I%)
                Percents(I%) = (SalArray(1) / Total) * 100
                LPRINT TAB(12); Range$(I%);
                PercentOut = (SalArray(I%) / Total) * 100
                LPRINT TAB(38);
                LPRINT USING "###"; SalArray(I%);
                LPRINT TAB(68);
                LPRINT USING "##.#"; PercentOut
                LPRINT
                LPRINT
                LPRINT
```

```
        NEXT I%
RETURN
DATA "< $25000","$25000 - 40000","$40000 - 50000","> $55000"
```

图 10.1(a) 例 2 的程序

```
001,"Paul Newman",77000
002,"Linda Evans",33000
003,"Mickey Mouse",44000
004,"Donald Duck",58000
005,"Nancy Stern",42000
006,"Robert Stern",38000
007,"Michael Douglas",22000
008,"Robert Redford",57000
```

图 10.1(b) 输入样例

```
                              SALARY REPORT

SALARY RANGE        TOTAL NO. OF EMPLOYEES        % OF EMPLOYEES

  $25000                    1                          12.5

 $25000 - 40000             2                          25.0

 $40000 - 50000             2                          25.0

 > $55000                   3                          37.5
```

图 10.1(c) 输出样例

自我测验(三)

1. 上例1中，销售数据是从键盘输入的．假定数据从磁盘文件读入，格式为：SalespersonNo，TransAmt，改变例 1 程序使它能处理磁盘文件销售量记录而不是处理键盘输入的销售量记录，文件中推销员序号是无序的．
2. 假定上述输入文件销售人员的序号是有顺序的，使用控制中断处理技术重编该程序．在问题 3、4 中，使用了由 DIM SalerTotal（1：25）定义数组 Sales Total,假定数据已在数组中，编制一段程序．
3. 打印总销售额超过 $ 5000的销售员人数．
4. 打印销售量最多的销售员．

5. 请看下面的输入:

```
ENTER SALES AMOVNT
?
ENTER MONTH OF SALES 1-12
?
```

销售量同月份一起录入，销售量为当月销售，假定所有数据都用上面那一年的数据，按月份打印出报告:

```
Month          Total Sales
01             $XX, XXX.XX
 .                  .
 .                  .
 .                  .
12             $XX, XXX.XX
```

6. 假定读入下面的数据:

```
Enter Day of Year (1–365):
Enter Daily Double Combination:
```

把这些数据输入一个数组，则当DailyDoubleArray(1) = 45时，4号和5号赢得元月一日的加倍.

编制一读入数据的程序，确保数据是有效的. 也即，所有1～365的元素都应该有值. 此外，在数组中存储的两位数，每一位都应取 1～8 的值；因为每次比赛有8 匹马. 如数据无效，打出错误.

7. 使用上面的数据，打印出一年内6–3结合的马赢得加倍的次数.

8. 写出建立25个数组元素的程序，按如下方式打印出来:

```
Element       Value
   1           XX
   2           XX
   3           XX
   4           XX
   .           .
   .           .
   .           .
  25           XX
```

9. 编写如同题8一样打印输出同一数组的程序，但顺序相反(如下所示):

2A8-32

```
Element          Value
  25              XX
   .               .
   .               .
   .               .
   4               .
   3               .
   2               .
   1              XX
```

10. 编写计算打印上面数组总和及均值的程序，假定数据已录入。

§10.5 数组元素排序

§10.5.1 替换方法

先放置10个元素的数组，然后按升序排列；示例中使用了10个元素，但也可是任意数目元素。该程序要求：

1. 按常规方法用FOR … NEXT循环存储数组元素，并用INPUT、READ语句。

```
OPTION BASE 1
DIM Array1(10)
FOR I% = 1 TO 10
    READ Array1(I%)
NEXT I%
.
.
.
DATA 85, 62, 51, 97, 73, 66, 24, 45, 37, 22
```

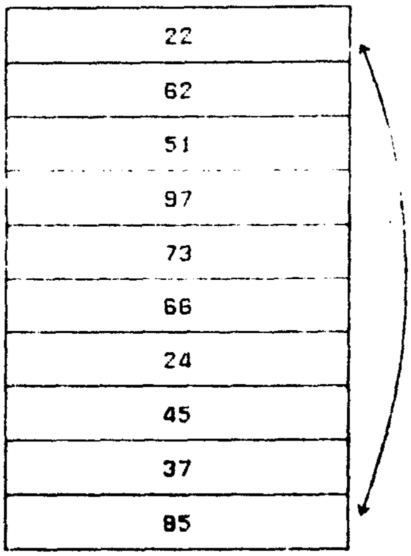
数组应存储如下：

85
62
51
97
73
66
24
45
37
22

2. 求出该数组中最小值；它可由下面程序完成:

```
Minimum = Array1(1)
FOR I% = 1 TO 10          'This could also be 2 TO 10
    IF  Array1(I%) < Minimum THEN
        Minimum = Array1(I%)
    END IF
NEXT I%
```

3. 重新编排这一数组，使得第一个元素是最小的一个，紧接是次最小者，依此类推。程序员必须找出上面的最小元素，并把它放到 Array1 (1) 处。原来的 Array1 (1) 同新的 Array1 (1) 交换。也就是说，不仅要把 Array1 (I%)移动至 FOR … NEXT 前面的最小位置上，而且要保留最小值的下标；例如: Array1 (10) 是最小的元素，把它同 Array1 (1) 交换，则数组变成如下形式:



(a) 22 is the minimum value.

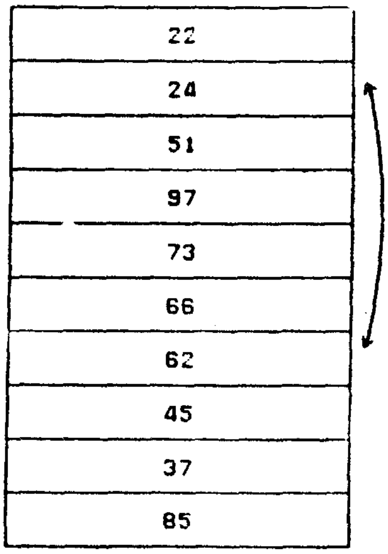
(b) It was originally the tenth element.

(c) The first and tenth elements are swapped.

要完成这一过程，前面的 FOR … NEXT 必须修改如下：

```
Minimum = Array1(1)
FOR I% = 1 TO 10
     IF  Array1(I%)   Minimum THEN
         Sub1 = I%
         Minimum = Array1(I%)
     END IF
NEXT I%
'To swap the two elements, code
Temp = Array1(1)
Array1(1) = Minimum
Array1(Sub1) = Temp
```

4. **前面的程序把最小值放到Array1（1）位置上，并同原来的array1（1）交换。要对数组进行全部排序，必须分别对 Array1（2） … Array1（10）重复这一过程；Array1（2）将成为次最小值的位置，原来的 array1（2）元素同次最小元素交换。**



These two elements are now swapped

紧接着对Array1 (3)，Array1 (10) 重复这一过程，等等。

要完成整个过程，程序前面的下标 1 应该用变量取代；下标从 1 开始，然后变为 2，等等。

```
'************************************************
  The FOR J% ...NEXT J% is the major loop
************************************************
FOR J%  = 1 to 10
        Sub1 = J%
        Minimum = Array1(J%)
        FOR I% = J% TO 10
                IF Array1(I%)   Minimum THEN
                        Sub1 = I%
                        Minimum = Array1(I%)
                END IF
        NEXT I%
 ************************************************'
  The follow three lines swap the two elements -
  the minimum value with where it belongs in the
  array
'************************************************
        Temp = Array1(J%)
        Array1(J%) = Minimum
        Array1(Sub1) = Temp
NEXT J%
```

图 10.2 所示为读入 10 个数组元素并按升序排序，依次从 1–10 打印结果的完整程序。该程序可经修改，以适用于任意多的元素。

```
  This program sorts the 10 elements in an array
CLS
'************************************************
' The following module defines and loads the
       table
'************************************************
OPTION BASE 1
DIM Array1(10)
FOR I% = 1 TO 10
        READ Array1(I%)
NEXT I%
'************************************************
'  The FOR J% ... NEXT J% is the major loop
'************************************************
FOR J% = 1 TO 10
        Sub1 = J%
        Minimum = Array1(J%)
        FOR I% = J% TO 10
                IF Array1(I%) < Minimum THEN
                        Sub1 = I%
                        Minimum = Array1(I%)
                END IF
        NEXT I%
'************************************************
' The following three lines swap the two elements -'
' the minimum value with where it belongs in the
'     array
'************************************************
        Temp = Array1(J%)
        Array1(J%) = Minimum
        Array1(Sub1) = Temp
NEXT J%
'************************************************
'The following coding prints the array elements in
' ascending sequence from 1 to 10
'************************************************
```

```
FOR I% = 1 TO 10
        PRINT Array1(I%)
NEXT I%
'**************************************************
' Data follows
'**************************************************
DATA 85,62,51,97,73,66,24,45,37,22
END
```

图 10.2(a)　读入排序打印 10 元素的程序

```
22
24
37
45
51
62
66
73
85
97
```

图 10.2(b)　上述程序的输出

注：Swap 语句用来交换两个变量的内容.

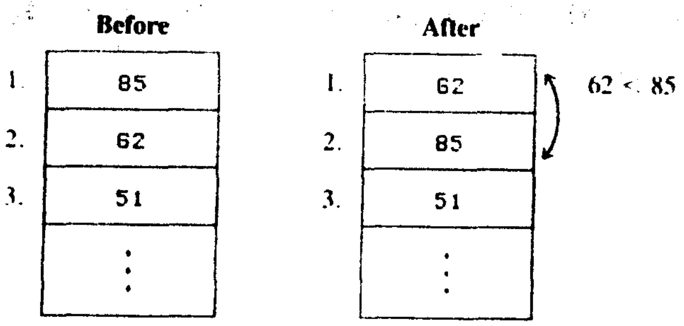
例:

SWAP Array1 (b1), Array1 (1)

§ 10.5.2　冒泡排序

用数组对元素排序的第二种方法叫冒泡排序 (bubble sort). 这里, 用到对头两个元素相比较的技术. 如果第一个元素小于第二个元素, 则不必要交换, 否则就要进行交换.

例:



对元素 2 和 3 也进行比较, 进行同样的操作; 即, 如第 2 个元素小于第 3 个元素不作交换, 否则亦交换.

	Before		After
1.	62	1.	62
2.	85	2.	51
3.	51	3.	85
	⋮		⋮

51 < 85

这一过程反复进行，直到各对元素都作过比较

Initial Values	Stored Values After First Pass
85	62
62	51
51	85
97	73
73	66
66	24
24	45
45	37
37	22
22	97

整个过程重复进行，先由元素 1 和 2 比较，紧接着元素 2 和元素 3 比较，依次比较下去。

第二遍比较之后的排序值:

51
62
73
66
24
45
37
22
85
97

这一过程反复进行，直至所有的值都按序存放，才不再进行交换。图 10.3(见下页)是对已储存了数的 10 个数组元素进行排序的程序段，这个过程就叫做冒泡排序。

注意：初始化把SwitCh $ 设置成"ON"状态；每次经过DO WITILE … LOOP把开关为"OFF"状态，但是如果进行了交换，则又把它置成"ON"状态；也即，SwitCh $ 在 DO WHILE 循环中，只要没有对数组进行交换就保持"OFF"状态。当 Switch $ 保持在"OFF"状态时，就可认为数组排序已结束。

DO WHILE 结构中的 FOR … NEXT 循环对相邻两个元素进行比较，如果第一个大于第 2 个则交换。

下面的程序对两元素进行交换。

```
Temp = Array1(I%)
Array1(I%) = Array1(I% + 1)
Array1(I% + 1) = Temp
```

```
'****************************************
' This program illustrates a bubble sort
' procedure
'****************************************
CLS
OPTION BASE 1
DIM Array1(10)
FOR I% = 1 TO 10
        READ Array1(I%)
NEXT I%
DATA 85,62,51,97,73,66,24,45,37,22
'****************************************
' Bubble sort procedure goes here
' Switch$ is set to "ON" initially - it
'    keeps track of whether any changes
'    have been made - when Switch$ is at
'    OFF at the end of a cycle, then no
'    changes were made and the array is
'    completely sorted
'****************************************
Switch$ = "ON"
DO WHILE Switch$ = "ON"
        Switch$ = "OFF"
        FOR I% = 1 TO 9 'FOR ...NEXT upper limit is 10 - 1
                IF Array1(I%) > Array1(I% + 1) THEN
                        Temp = Array1(I%)
                        Array1(I%) = Array1(I% + 1)
                        Array1(I% + 1) = Temp
                        Switch$ = "ON"
                END IF
        NEXT I%
LOOP
FOR I% = 1 TO 10
        PRINT Array1(I%)
NEXT I%
PRINT "Array has been sorted"
END
```

图 10.3(a) 使用冒泡排序法的程序段

```
 22
 24
 37
 45
 51
 62
 66
 73
 85
 97
Array has been sorted
```

图 10.3(a) 程序的输出样例

§ 10.6 数组处理的几个补充概念

§ 10.6.1 界限选项

在编制使用数组程序以前，应该设置界限选项（Bounds option）。如果 Bound 可选项为 ON，计算机将检查所有的下标是否有效。下标检查可确保该下标在数组定义的范围内；如果程序使用了太小或太大的下标，并当 Bounds 选项为 ON 时，将出现下标范围错误信息（错误 9）。请看下面的程序：

```
OPTION BASE 1
DIM Array1 (25)
FOR I% = 1 TO 35
    INPUT Amt1
    Array1 (I%) = Amt1
NEXT I%
```

该程序在 FOR … NEXT 循环中有错误，应该用 FOR I% = 1 To25 而非 35；数组说明是 25，故下标变量 I%应从 1–25 变化。

如果 Bounds 可选项不在 ON 状态，上面程序就会引起运行错误。确切地说，多余的 10 个输入量将取代内存中其它输入量，极有可能把程序中的数据全毁掉。由于部分内存区域被 10 个附外量所取代，该程序可能不起作用。这个程序将产生不可预测的结果，且难于调试。

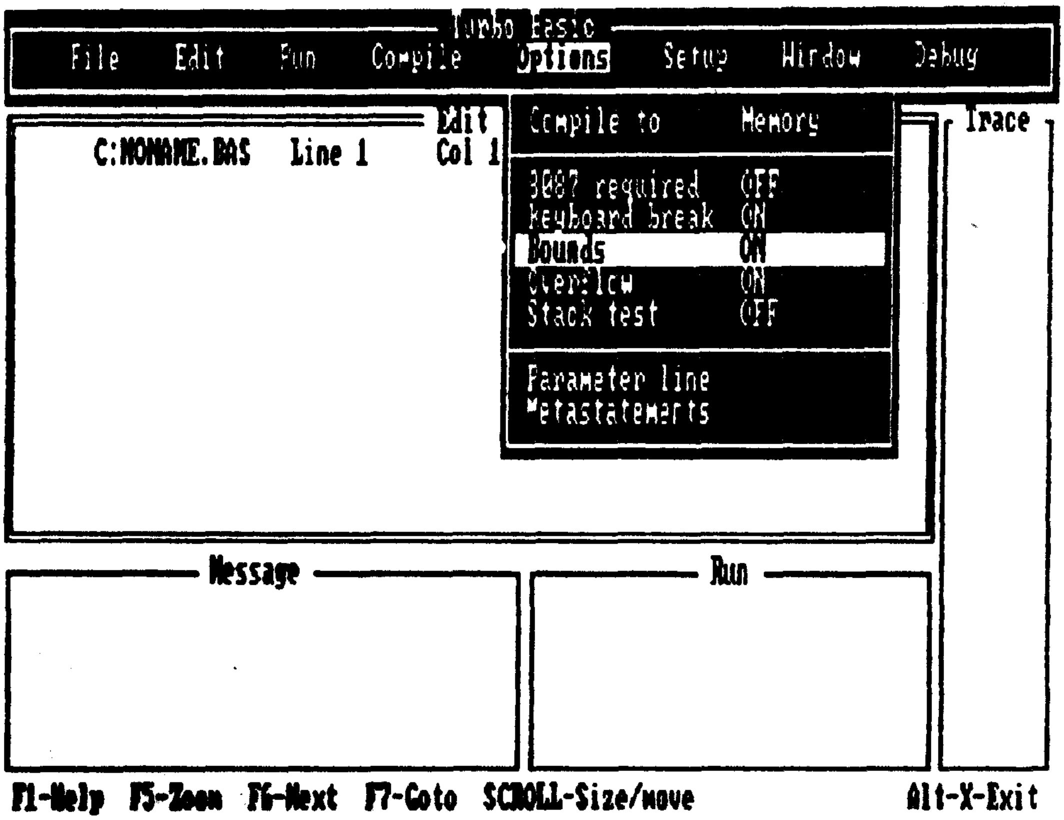
如果把 Bounds 选为 ON，则读入第 25 个元素后就会出错，这一错误表明 FOR … NEXT 循环中出现下标错误。

Error 9: Subscript out of range

把 Bounds 选为 ON，计算机就检测下标错误,具体步骤如下：

1. 在主选单中，用→键，把光标移到Options命令上，然后按回车。
2. 按"↓"键三次，使Bounds选项变为高亮度。
3. 按回车键把Bounds由OFF改变成ON状态。
4. 按Esc键，脱离Options选单。
5. 运行时，Bounds就设置为ON状态；如要长期保留这一状态：
   a. 用箭头"→"键，把Setup选项选为高亮度。
   b. 按"↓"键四次，光标到Save Options / window / setup，再按回车，计算机要求用户确认这一更改。然后把 Bounds ON 选项存到配置文件长期保存。

注意：Bounds ON 选择可确保只能使用有效数据做下标。



考虑下面的程序，假定数组 TotalArray 的元素已初始化为 0 值.

```
DIM TotalArray (1:50)
.
.
.
INPUT SalesNoIn%
INPUT Amt
LET TotalArray (SalesNoIn%) = TotalArray (SalesNoIn%) + Amt
```

该程序只要 SalesNoIn%取 1−50 的值，则正确运行；如果你没有设置有效化过程，且 SalesNoIn%小于 1 或大于 50，那么超出数组界限的其它存储区将由 LET 语句来改变．最糟的是，计算机不告诉用户有错误发生，除非事先把 Bounds 选项设为 ON．

总的来看，定界错误可能导致输入错误、不可预见的错误处理和运行错误．为了确保这一错误不致发生，可编制前节所述的数据确认程序或把 Bounds 选项置成 ON，或两者皆做．如果把 Bounds 选为 ON，则任一小于 1 或大于 50 的下标值都将引起错误信息显示．

§ 10.6.2 动态及静态数组

静态数组要求有确定的元素个数，不用 DIM 语句是不能改变其大小的．例如，要打

印每周的日销售量，那么数组元素的个数就定为5，即每个工作日算一天.

```
OPTION  BASE  1
DIM  DailySalesTotal (15)
```

这种数组就叫静态数组. 然后，如果希望把一周7天的总销售量记录下来，就得把DIM语句变成如下形式:

```
OPTION  BASE  1
DIM  DailySalesTatal (7)
```

现在，假定要建立关于每个客户成交总额的数组；客户的总数随时都在变，那么建立一个DailySalesTotal数组，其元素个数在程序运行时决定而不是一次编程完成，这样效果会更好. 我们称其元素大小可变，且在运行中确定的数组为"动态数组".

缺省时，Turbo Basic默认数组为静态数组. 这个缺省值可由$STATIC元语句显式指定. 元语句（Metastatment）是一条编译指令（compiler directive）. 它的功能不同于标准语句，标准语句在程序运行时执行. 而元语句则在编译时控制编译，元语句以美元符号（$）打头，易于在程序中设置. 此外，每行只能有一条元语句. $STATIC元语句可显式定义数组为静态数组，即数组元素个数固定. 要显式定义所有数组为静态数组，可编程如下:

```
$STATIC    ┐
   :       │
   :       ├  此处全部数组都将为静态数组
   :       │
           ┘
```

正如上面所述，如果省略$STATIC元语句也完全可得到同样的效果，因为缺省设置为静态数组.

但是，如果要定义动态数组，则必须用$DYNAMIC元语句；它表示在程序进行时，可设置动态数组的大小.

对动态数组，可用DIM语句或用大小可变的变量来规定它的上限.

例1:

```
$DYNAMIC
INPUT N
DIM ARRAY1 (1:N)
'The size of the array will be set by the user
```

例 2:

```
$DYNAMIC
DIM ARRAY2 (1:100)
'The maximum size of the array is 100 elements
```

一旦规定了上限，无论是变量形式还是常数形式，数组界限就将同静态数组一样；如果 Bounds 可选项为 ON，则进行越界检查.

例 3: 把全部数组定义为动态数组.

```
$DYNAMIC
:
DIM Array1 (1:10)       'The maximum size of Array1 is 10
DIM Array2 (1:25)       'The maximum size of Array2 is 25
```

例 4: 定义 ArrayA 为动态数组，定义 ArrayB 为静态数组.

```
$DYNAMIC
'All arrays from this point on will be dynamic unless or until
'a $STATIC metastatement is executed.
:
DIM ArrayA (1:50)
:
$STATIC
'All arrays from this point on will be STATIC unless or until
'a $DYNAMIC metastatement is executed.
:
DIM Arrayb (1:10)
:
```

例 2 中，如果省略 $ STATIC 元语句，ArrayB 将被当作动态数组来用，因为 $ DINAMIC 元语句针对其后的全部数组.

我们也可把 DIM 同 STATIC 或 DYNAIC 放在一起当作一个记号.

例:

```
1. DIM DYNAMIC ArrayA (1:50)
   DIM STATIC ArrayB (1:10)
2. $DYNAMIC
   INPUT N
   IF  N = 1 THEN
       DIM ArrayX (1:10)
   ELSE
       DIM ArrayX (1:50)
   END IF
```

前面已提到，ERASE 语句可把静态数值数组置为 0，把静态字符串数组置成空。但 ERASE 语句对动态数组则有不同的效果，它将把动态数组全部删除。这意味着起初给动态数组所分配的内存空间经 ERASE 后，可供其它处理使用。

用 ERASE 语句删除动态数组，对于大型程序来说是非常有用的，因为程序有可能超出所配给的存储空间；这样，我们先处理数组，然后将它删掉，那么它所占据的存储空间就可另作它用。但是，要注意，删除这些动态数组时，一定要确保后面的程序不用到这些数组。

本章将继续讨论静态数组；通常，当程序太大计算机不能处理时，倾向于使用动态数组。

§ 10.7 使用数组存储表格数据

前面，我们集中讨论了用数组储存总和及建立以整体看待的一系列数组元素。这一节将介绍用数组存储表格数据。表格（table）就是由一系列数据构成的清单。在程序处理过程中需要某特定的项时，可参考或查询表格；税务表和帖现率表就是计算机储存表格的例子。当处理细目时，"查表"以作参照。

例 1:

假定一邮购公司要把帐单送到美国各地的私人手里，需设计这样一个程序：(1)读入顾客含有帐单情况的输入数据；(2) 在帐单实际运送时，输出顾客帐单；因为美国各个郡都有不同的税收制，因此还必须有计算销售税的程序。处理这些帐单要用到几种不同的方法。

1. 应把实际税率作为每个输入记录的一部分。

   在每个输入记录中都键入税率是非常麻烦的；首先，税率时有改变。每当一个郡的税率发生变化时，有关这个郡的所有输入记录都得改变。这可能是一个费时又麻烦的工作；其次，对每个输入记录都输入税率意味着额外的数据录入，如此给用户带来额外的劳动量且输入数据出错的危险更大。

2. 把税率作为一个表格输入，需要时再引用。

这个方法在存储税率上比第一个方法要效率高得多，效果又好。程序的输入应该由两个独立的数据元素组成。在记录输入之前，就应该录入每个郡相应的税率表，然后存储起来。如果这个表不太大，可以从键盘录入；但若这个表较大，就该从顺序文件中接收数据。因为数据是作为输入读入的，所以这种表格称作"外部表格（external table）"；税率也可用 READ 和 DATA 语句读入，当需要时引用。这种表格称为"内部表格（internal table）"，因为数据存在程序中。

表格读入后，应该输入销售文件；对每个输入记录，都该用这个表来查找其相应郡县的税率。

§ 10.7.1 并行数组

用两个一维数组定义一个表格，这样的数组叫"并行数组（parallelarrays）"。到目前为止，我们主要讨论了数组存储总和及存储一系列元素的用途，元素要么全是数字式的，要么是字符串式的。这一节将集中介绍存储表格数据的两个独立并行数组：一个数组包含有表格变元（table argument），诸如郡和划分美国邮区的五位号码；另外一个数组则包含表格函数(table function)，其中有所要查找的数据；本例中即为税率值。

我们把两个并行数组定义成一个表：一个数组含有表格变元，另一个数组包含有相应的并行表格函数值；即，如果第一个数组中 TaxDistrict5 是我们给定的输入记录的表格变元，那么在第二个数组中第五个税率就应该是想要查找的税率或表格函数。

我们已注意到表格数据可按数字形式输入到程序中。如果这个表是个小表格，可通过键盘间断录入这些数据，也可从一个生成的单独文件中读入这些数据；这个文件要靠单独的程序来维护。如上面所述，作为输入录入的表格叫外部表格。表格也可以用 READ 和 DTAA 语句存储成为内部表格；无论哪种情况，程序中都应该把表格以两个并行数组方式存储。

§ 10.7.2 把数据存入表格

把税率存入由两个并行数组组成的表格效率更高，不仅是因为这样减小了键盘输入和数据录入操作，而且税率在一个表里更加易于维护、修改；即如果税率有变，只需要改变与那个税率有关的单个表格就行。如果这个表格是个独立的文件，可单独编程处理之。如税率作为内部表格储存，只需改变相应的 DATA 语句，但是如果税率数据包含在输入的顾客数据中，则需修改由税率变化所涉及到的所有顾客的记录。假定有 1000 种税率，10000 个顾客帐目，要把税率同每个输入放在一起作为其中的一部分输入，则需要输入 10000 个附加变量；但把 1000 个税率作为一个表存储效率就高得多。如前所述，完成这一工作可通过下面任一方式。

(1) 把这1000个税区数据录入（从键盘或从文件）到一个数组当中，再把1000个税率放入一并行数组中。

(2) 使用READ或DATA语句存储这些数据，然后便可在每个录入的顾客帐目上，查找到存储在表格中的相应的税率。表格由两个数组构成。

存储表上，应该把税率同每个特定的税区联系起来；我们将使用五位数的邮政编码，作为表格变元来区分每个税区。

假定税率表里把五位的邮政编码作为表格变元放到一个数组里，而把其相对应的税率作为表格函数放到另一单独的数组中；在查找税率时，将把每笔交易记录的五位邮政编码同数组中的邮政编码相比较。

输入五位邮政编码：12311		五位邮政编码数组	销售税率数组
	(1)	XXXXX	.XXX
	(2)	XXXXX	.XXX
		·	·
		·	·
		·	·
		12311	.XXX
		·	·
		·	·
		·	·
	(15)	XXXXX	XXX

§10.7.3 找出与销售输入相匹配的表格条目

表格里的一系列输入数据（五位邮政编码和税率）首先要从键盘上输入，或是从一个文件上读出；亦或是用 READ 和 DATA 语句生成。然后，程序把五位邮政编码和税率存储到两个单独的并行数组中。之后，读入顾客交易记录，每笔交易都该包含顾客的五位邮政编码、定购量和单位价格。这些交易数据可放到磁盘文件中，或者用键盘间歇地输入。对每笔交易，都要把顾客的五位邮政编码同储存在表格里的五位邮政编码相比较，直到找到相匹配的编码为止。这个用于比较的顾客的五位邮政编码称作查询变元。

在寻找相匹配的编码时，下标值指示 zipcode 数组中相匹配的五位邮政编码。可用该下标把相应的税率从 SalesTaxRate 数组中读出来。然后用这一下标所得到的税率来计算总的价格。

假定这两个数组是 zipcode（1：15）和 SalesTaxRate（1：15），当 zipcode（I%）= CustomerZipcode 时，SalesTaxRate（I%）就是我们想要得到的并行数组中的相应税率。

编制理想顾客帐单的 Turbo Basic 程序分两个步骤：

1. 以并行数组的方式读入并存储表格。表可从一个顺序文件中读入也可从键盘式输入，但都是外部表格；或者以内部表格的形式用 READ 或 DATA 语句存储数据表格。
2. 从磁盘上或在终端上处理每个输入记录，这个过程包括表格查找；即把每个Cus-

tomerCode 同数组中的 zipcode 相比较.

假定有 15 个如下形式的表格数据:

ZipCode:
SalesTax:

表格数据可从键盘输入, 用下面方法读入表格:

```
DIM ZipCode(1:15)
DIM SalesTaxRate(1:15)
FOR I% = 1 TO 15
    PRINT "Enter Zip Code:"
    INPUT ZipCode(I%)
    PRINT "Enter Sales Tax:"
    INPUT SalesTaxRate(I%)
NEXT I%
```

使用 READ、DATA 语句, 建立表格程序如下:

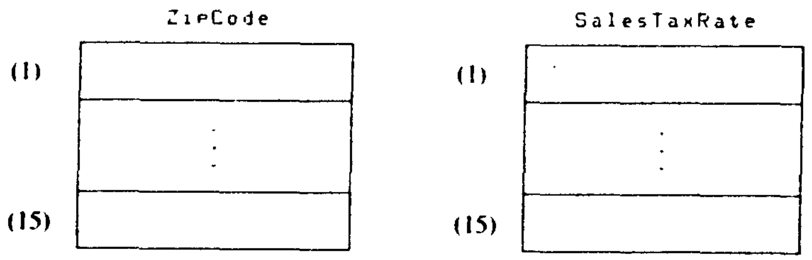
```
DIM ZipCode (1:15)
DIM SalesTaxRate (1:15)
FOR I% = 1 TO 15
    READ ZipCode(I%)
    READ SalesTaxRate(I%)
NEXT I%
'Data consists of a zipcode
'followed by a tax rate. There are
'15 pairs of zip codes and tax rates
DATA 11027, .07, 11532, .06, ...
```

最后, 假定将 15 对五位邮政编码和税率数据存放到输入顺序文件中, 并用一单独的程序维护. 我们可用下面程序从这个文件中读入该表.

```
OPEN "Table.Dat" For INPUT AS #1
DIM ZipCode (1:15)
DIM SalesTaxRate (1:15)
FOR I% = 1 TO 15
    INPUT #1, ZipCode(I%), SalesTaxRate(I%)
NEXT I%
```

表格数据是从键盘交互输入, 还是作为外部表格从一个顺序文件读入, 亦或作为内部表格用 READ、DATA 语句读数据, 取决于表格的应用情况及大小. 键盘输入一个大表格是不实际的; 用 READ、 DATA 语句储存经常变更的大表格也是不妥当的. 生成并维护一个顺序表格文件是处理频繁变化、大表格的最好选择.

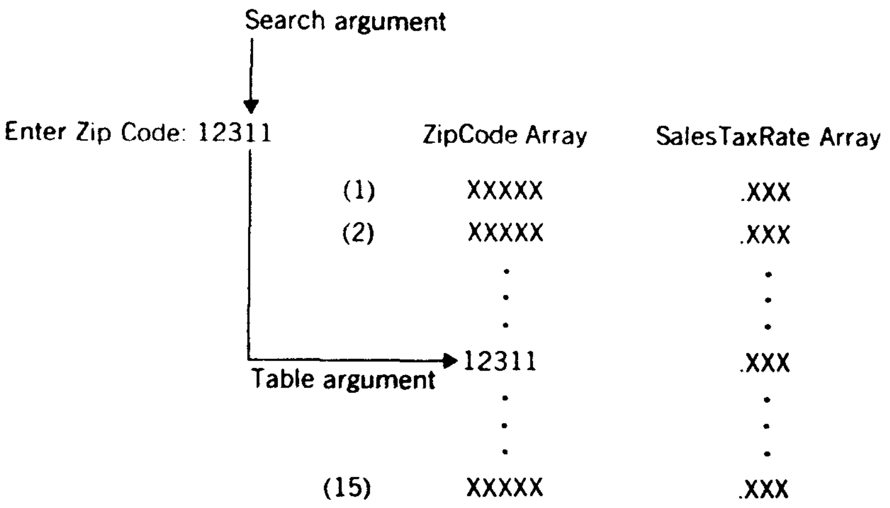
下面生成用作表格的并行数组:



每笔输入交易处理之前，要先把表格数据读到并行数组中；这样，由这两个并行数组构成的数据表就不必象储存总和数值那样要进行零值初始化。储存总和的数组是要对它们累加，而表格是要读入或输入数据的。

表格数据存储好后，就应读入顾客的帐单数据；然后显示输出每个顾客包括税在内应收的总费用。要找到每个顾客的税率，就得查找表格中的五位邮政编码，直至找到与顾客的邮政编码相匹配的码为止。在寻找匹配的邮政编码时，我们再用相同的下标来查找另一个并行数组 SalesTaxRate 中的相应税率。

然后，就可用这个税率来计算出每个顾客应收费的总额这个总额中包含销售税。正如前面提到的，表格变元是 zipcode 数组中的变量，要同顾客数据比较；要查表的元素，本例中为税率，称作"表格函数 (table function)"；用来寻找匹配值的交易数据称作"查询变元 (seareh argument)"。



计算单个输入销售税的程序如下:

```
LET I% = 1                              'Initialize the subscript
PRINT "Enter the Unit price and Quantity"
INPUT UnitPrice, Qty
PRINT "Enter ZipCode'
INPUT ZipIn
DO  WHILE ZipIn <> ZipCode (I%)        'Search the table
    LET I% = I% + 1
LOOP
LET Total = (1 + SalesTaxRate (I%)) * (UnitPrice * Qty)
PRINT "Total is "; Total
```

下标I%初始化为1；为了强调下标必须是整型数，我们使用I%而不是I。

§10.7.4 找不到匹配编码怎么办

正常情况下，执行DO WHILE … LOOP循环，当找到匹配值时退出。但是，我们必须考虑这个匹配值找不到的可能性；也就是说，必须设计一段程序，当交易的五位邮政编码同表格中的编码不匹配时，打印出错信息；程序如下：

```
LET I% = 1
PRINT "Enter the Unit price and Quantity"
INPUT UnitPrice, Qty
PRINT "Enter Zip Code"
INPUT ZipIn
DO  WHILE I% <= 15 AND ZipIn <> ZipCode (I%)
    I% = I% + 1
LOOP
IF  I% > 15 THEN
    PRINT "No Match Found"
ELSE
    LET Total = (1 + SalesTaxRate (I%)) * (UnitPrice * Qty)
    PRINT "Total is "; Total
END IF
```

DO WHILE … LOOP循环直到找到了匹配编码，或表格全部查询完毕才终止执行。每种情况下，DO WHILI … LOOP后的指令都要执行。如果匹配编码找到了，下标I%的值就指向ZIPIN和Zipcode（I%）之间的匹配编码；如果找不到匹配编码，就有可能是顾客的五位邮政编码输入有错，也可能是表格的邮政编码有错。此时显示错误信息，如果不进行错误测试，就会得出不正确的输出结果，甚至会出现运行错误；简言之，在找不到匹配时必须保证表格数据已全部查询完，循环才结束。也就是说，要保证不发生运行错误、无限循环或结果错误，就得当I%超过15，查询终止。请记住，尽可能让计算机发现错误才是最好的方式，所以要把Bounds选项设为ON。

例2　假定对在某指定商店购买东西的顾客给予帖现；现有500个顾客，但顾客序号是不相连的。按下面方式读入两个并行数组：一个数组放顾客的序号；另一个数组放相应顾客的帖现率。这样，由这两个数组形成一个表。

顾客序号 (表格变元)		贴现率 (表格函数)
[1]	001	.03
[2]	003	.01
[3]	004	.05
[4]	007	.07
⋮	⋮	⋮
[500]	982	.02

把这两个表格存起来后，就可读入顾客购物数据，并用并行表格录找出每位顾客的贴现率：

Enter Cust No: [   ] ←— Search argument

Enter Unit Price [   ]

Enter Qty [   ]

The (Gross Price – Discount) is: [   ]

计算新价格的程序如下：

```
CLS
DIM CustomerNo (1:500)

DIM DiscountPercent (1:500)
PRINT "TABLE ENTRIES FIRST"
FOR Count% = 1 TO 500
    PRINT "Enter customer number"
    INPUT CustomerNo (Count%)
    PRINT "Enter discount percent"
    INPUT DiscountPercent (Count%)
NEXT Count%
PRINT "CUSTOMER TRANSACTION DATA"
GOSUB GetCustomerData
DO  WHILE CustNoIn <> 0
    LET I% = 1
    DO  WHILE I% <= 500 AND CustomerNo (I%) <> CustNoIn
        LET I% = I% + 1
    LOOP
    IF  I% > 500 THEN
        PRINT "No match found"
    ELSE
        GOSUB CalculateRoutine
        PRINT "The (Gross Price - Discount) is :"; NewPrice
    END IF
    GOSUB GetCustomerData
LOOP
PRINT "End of Run"
END
```

```
GetCustomerData:
    PRINT "Enter Cust No: ";
    INPUT CustNoIn
    PRINT "Enter Unit Price: ";
    INPUT UnitPrice
    PRINT "Enter Qty: ";
    INPUT Quantity
RETURN
CalculateRoutine:
    GrossPrice = UnitPrice * Quantity
    NewPrice = GrossPrice - (GrossPrice * DiscountPercent (I%))
RETURN
```

注意：存储顾客的序号有必要用一个数组；贴现率也需要一个数组。将这两个数组结合起来，共同构成一个表。

自我测验(四)

1. 一个数组，其元素数目是可变的，大小在运行时确定，称作_____数组。
2. 如果想在运行过程中确定数组的大小，必须用_____元语句编程。
3. 写出几条指令，定义含有100个元素的数组computers为静态数组，定义最大不超过 150 个元素的数组 NOROOMS 为动态数组，定义 Classes 为无上界动态数组。
4. （TorF）不论是动态数组还是静态数组，ERASE语句都可把任何数字数组设置为零，把任一字符数组设置成空。
5. （TorF）表格变元可用来确定表格函数元素。
6. 编制对两个20个元素的并行数组进行定义、初始化及读入数据的程序，第一个数组放 Social Seurity 编号。第二个放 Emplyee 的名字。
7. 用第6题的表格，定也录入Socialsecurity编号及打印Employee名字的程序，假定表格已赋值，对每个输入的 Social Security 编号，打印出相应的 Employee 名字。
8. （TorF）读入表格数据之前，应对所有的输入交易进行处理。
9. （TorF）用来查找与表格匹配的交易数据称为查询变元。
10. 假定下面的并行数组已经赋值

```
DIM Model of Car$  (1 : 10)
DIM Price  (1 : 10)
```

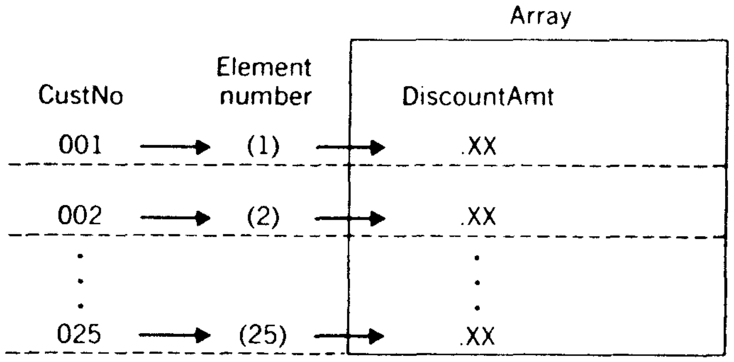
写出输入 Model of car $，并打印出相应价格的程序。

§10.7.5　根据表格中的相对位置寻找表格输入

如上所述，在大多数表格查询中，表格中的数据都含有象 ZipIn 或 CustNo 这些用来查询的表格变元；这些表格变元可能是，也可能不是顺序的，通常它们是不连续的。例如，在 CustomerDiscount 程序中的表格变元取 001、005、008 就是非连续的。但是，如

果表格变元是连续的（001、002、003 等），我们就可省略一个数组从而减少编程量；也就是说，表格可由一个数组构成。其 1 号元素指向 CustNo001，2 号元素指向 CustNo002, 依此类推；下面是单个数组构成的表格。

DiscoumtTable(不需要并行数组)



CustNo 的变化范围为: 001 ～ 025 —— 它可用作下标

该例中，数组元素 DiscountAmt（1）指向 CustNo001，DiscountAmt（2）指向 CustNo2, 依此类推。如果把 CustNoIn 作为一个输入变量读进来，那么就可用这个变量本身作下标，来找到相应的 DiscountAmt, 根本用不着查询；这是因为对于第 5 个 custNoIN，对应的 DiscountAmt 即为 DiscountAmt(5)。也就是说，CustNoIn 本身就可用作下标，没有必要把 CustNo 单独存入一个并行数组，因 CustNo 数组的第一个元素就应该是 001,第二个元素是 002, 等等。因此，存放顾客序号的数组是不必要的。对这个问题编程，要比需要两个并行数组时的编程容易得多。求各个顾客的贴现，可编程如下：

```
OPTION BASE 1
DIM DiscountAmt(25)
   .
   :            Routine to load table goes here
   .
INPUT CustNoIn
Total = UnitPrice * Qty
Net = Total - (DiscountAmt (CustNoIn) * Total)
```

要确保 CustNoIn 是小于 26 的数，就该对它进行检查。

对这个问题，不需要存储 CustNoTable，仅需存储 DiscountAmt；就是说，DiscountAmt（1）即指向 CustNo1，Discountamt（2）即指向 CustNo2，依此类推。图 10.4 是求解该问题的全部程序：

```
CLS
OPTION BASE 1
DIM DiscountAmt(25)
FOR I% = 1 TO 25
        PRINT "Enter discount for customer "; I%
        INPUT DiscountAmt(I%)
NEXT I%
PRINT "The discount table is full"
GOSUB GetCustomerData
DO WHILE CustNoIn <> 0
        IF CustNoIn < 26 THEN
                Total = UnitPrice * Qty
                Net = Total - DiscountAmt(CustNoIn) * Total
                PRINT "The (Gross Price - Discount) is ";
                PRINT USING "$##,###.##"; Net
        ELSE
                PRINT "Invalid Cust No "; CustNoIn
        END IF
        GOSUB GetCustomerData
LOOP
PRINT "End of Run"
END
GetCustomerData:
        PRINT "Enter Customer Number (Enter 0 if done)";
        INPUT CustNoIn
        PRINT "Enter Unit Price ";
        INPUT UnitPrice
        PRINT "Enter Quantity ";
        INPUT Qty
RETURN
```

图 10.4 表格处理程序

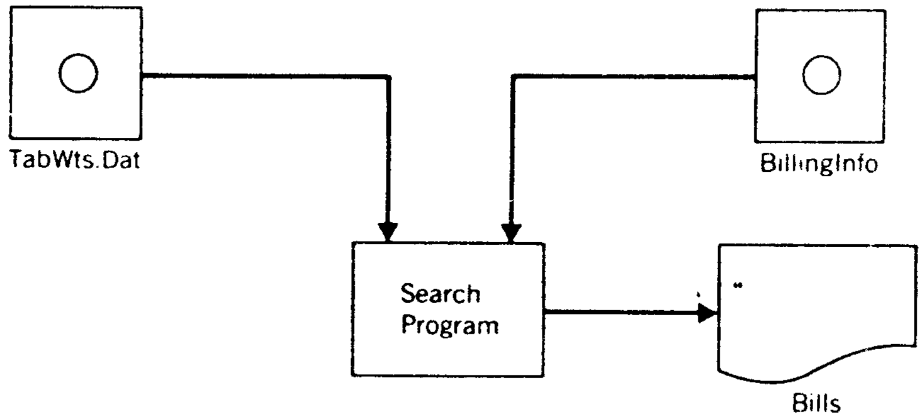
这个程序，省略了一个表格查询，只能适用于顾客序号从 1～25 接连排列的情况。例如，对第 22 个输入，相应的 CustNoIn 的确切值就是 22。因此，没必要把顾客序号 22 储存在一个单独的并行数组中。如果顾客序号是不连续的，就得按规定查询；即把顾客序号存入一个数组，把与之相应的 DiscountAmts 存入第二个并行数组中。前节例 2 就是并行数组用于表格查询的示例。

下一节，将示例说明数据、总和、表格是怎样用维数组存储的。

自我测验(五)

下图是针对问题 1－3

系 统 流 图



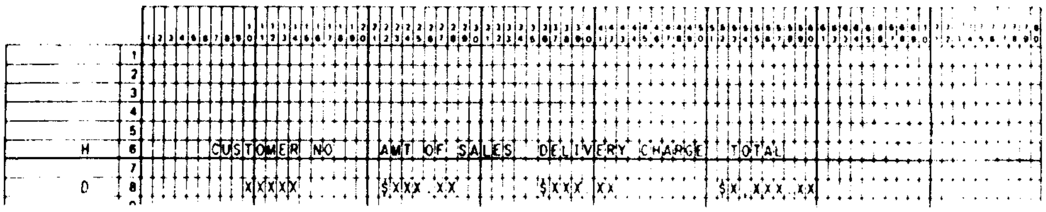
TabWts.Dat 记录格式(20 个条目)

WeightMax, DeliveryCharge

BillingInfo 记录格式

CustNo, Amt, WeightMarked

Bills 打印格式图



1. 编制对所需数组进行DIM说明的程序。
2. 编制读入表格数据的程序。
3. 编制处理顾客数据的程序。
4. 当一个数组的下标用来查询第二个数组中的数据时，这两个数组称为______。
5. (TorF) 所有的表格查询都应该确信：表格元素同交易文件中的数据间是相匹配

的。

下面是人口数组

```
OPTION  BASE  1
DIM  StatePop (50)
```

6. 写出输入50个州人口数字的程序，每行一个。
7. 求出U.S的总人口。
8. 求出人口超过10000000的州的个数.
9. 求出哪个州的人口数最多，是多少?
10. 求出人口等于1234000的州的个数.

§10.8 多维数组

§10.8.1 二维数组用于存储数据

到现在为止我们主要讨论了单维数组：即数组只有一个下标.

例1:

假定建立一个数组保存Los Angeles市全年每小时的温度数字，用单个下标就需要365个独立数组，每天一个；Array1的Temp (1) 元素存放元月1日上午1时的温度、Array1的Temp (24) 存放元月1日午夜的温度. 类似地，Array2的Temp (1) 储存元月2号上午1时的温度，依此类推；储存365天、每天24小时温度的另一方法就是：建立一含8760个元素的数组。元素1～24存放元月1号的温度，元素25～48存放元月2号温度，等等.

存储这些数据比较有效的方法是用两维数组，见下表。

LOS ANGELES TEMPERATURES

HOUR

		1 A.M.	2 A.M.	. . .	MIDNIGHT
DAY	Jan 1	(1, 1)	(1, 2)	. . .	(1, 24)
	Jan 2	(2, 1)	(2, 2)	. . .	(2, 24)
	⋮	⋮	⋮	⋮	⋮
	Dec 31	(365, 1)	(365, 2)	. . .	(365, 24)

这种类型的数组可用下面的方法建立。

Method 1

```
OPTION BASE 1
DIM Temp (365, 24)
```

Method 2

```
DIM Temp (1:365, 1:24)
```

在这些例子中均假定一年为 365 天，即不是闰年。OPTION BASE 1 指 Hour 从 1–24 变化、Day 从 1–365 变化。

Temp 数组需两个下标确定一个元素：一个下标指向日，另一个下标指向时。括号内的第一个下标是主下标，即下标指向日；第二个下标付下标为列下标，指向时。该数组有 8760 个元素，它们可分为 365×24 个输入项。

例如，Temp (1, 3) 存放元月 1 日下午 3 时的温度；Temp (2, 5) 存放元月 2 号上午 5 时的温度。括号内的数字指示相应的下标。

假定数组已输入数据，要求元月 5 日的平均温度；可编程如下：

```
CLS
OPTION BASE 1
DIM  Temp (365, 24)
Total = 0
FOR  I% = 1 to 24
     Total = Total + Temp (5, I%)
NEXT I%
Average = Total / 24
PRINT "The Average Temp for January 5 is "; Average
```

要求全年的平均温度，可用 FOR … NEXT 循环语句编程如下：

```
CLS
OPTION BASE 1
DIM Temp (365, 24)
Total = 0
FOR I% = 1 TO 365
    FOR J% = 1 TO 24
        Total = Total + Temp (I%, J%)
    NEXT J%
NEXT I%
Average = Total / 8760
PRINT "The Average Temperature for the Year is "; Average
END
```

注意： 对前面的程序也可先改变付下标 (Hoar)，后改变主下标 (Day)；也即，先把全年上午 1 时的温度累加起来，再把全年上午 2 时的温度累加起来，依此类推。这两个程序将产生相同的结果。

```
CLS
OPTION BASE 1
DIM Temp (365, 24)
Total = 0
FOR I% = 1 TO 24
    FOR J% = 1 TO 365
        Total = Total + Temp(J%, I%)
    NEXT J%
NEXT I%
Average = Total / 8760
PRINT "The Average Temperature for the Year is "; Average
END
```

读入数组值可如下编程:

To load the array, we may code:

```
OPTION BASE 1
DIM Temp (365, 24)
FOR I% = 1 TO 365
    FOR J% = 1 TO 24
        PRINT "Enter Temp for Day "; I%; " Hour "; J%
        INPUT Temp (I%, J%)
    NEXT J%
NEXT I%
```

或者，用 READ、DATA 语句读入数据:

```
                  OPTION BASE 1
                  DIM Temp (365, 24)
                  FOR I% = 1 TO 365
                      FOR J% = 1 TO 24
                          READ Temp (I%, J%)
                      NEXT J%
                  NEXT I%
                  'Each Data statement would have 24 values
                  'The first Data statement relates to Day 1,
                  'The second to Day 2, etc.
                  DATA 73, 24, 66, ...         '24 entries here
365             } DATA ...
DATA            }   .
statements      }   .
                    .
```

READ 和 DATA 语句最好用来给大表格赋值；如要求用户输入 8760 个数据，出错的危险性是很大的。因此，如果输入一个错误元素，整个数据输入过程就得重复。

求出一年的最高温度

```
'Assume the array has already been loaded
'Program excerpt finds highest temperature
CLS
OPTION BASE 1
DIM Temp (365, 24)
.
.          'Array is loaded here
.
Highest = Temp (1, 1)
FOR I% = 1 TO 365
    FOR J% = 1 TO 24
        IF  Temp (I%, J%) > Highest THEN
            Highest = Temp (I%, J%)
        END IF
    NEXT J%
NEXT I%
PRINT "Highest Temperature Is "; Highest
END
```

求温度底于 50℃的天数

```
'Assume the array has been loaded
OPTION BASE 1
DIM Temp (365, 24)
Counter = 0
FOR I% = 1 TO 365
    FOR J% = 1 TO 24
        IF  Temp (I%, J%) < 50 THEN
            Counter = Counter + 1
        EXIT FOR
        END IF
    NEXT J%
NEXT I%
PRINT "No of Days when Temperature < 50 was "; Counter
END
```

注意：上面的程序遇到温度低于50就退出付循环。当遇到温度低于50时，不需检查那天其余的温度，所以用 EXIT FOR 退出付循环。

例 2:

考虑下面的二维数组，假定数据已读入。

Method 1

```
OPTION BASE 1
DIM PopulationFigures(50, 10)
```

Method 2

```
DIM PopulationFigures(1:50, 1:10)
```

PopulationFigure 数组定义了 500 个变量，即把 50 个州的每一个均分成 10 个郡；其人口数字如下：

COUNTY

STATE	1	2	3	4	5	6	7	8	9	10
1										
2										
⋮ 50										

DIM Population Figures（50，10）语句建立包含 50 主组也即 50 行（州）的数组，每得又分为 10 个付组也即 10 列元素（郡）。

假定数据已输入数组，就可把它累加起来成为美国的总人口，即把每个州的 10 个郡的数字加起来。把 Populationfigures（1，1），（1，2），… （1、10）加起来，可得到 Statel 的人口；把 Population Figures（2，1），（2，2），… （2，10）累加起来，可得到 State2 的人口，依此类推。从主下标 1 开始，把付下标从 1～10 变化；然后把主下标加 1，付下标再从 1～10 变化，一直进行下去，直到每个州中 10 个郡的人口都已累加为止。

该程序需用嵌套的逻辑控制结构。FOR … NEXT 循环语句能自动增加下标，用作嵌套循环是最合适的。

```
USATotal = 0
OPTION BASE 1
DIM PopulationFigures(50, 10)
FOR StateSub% = 1 TO 50
    FOR CountySub% = 1 TO 10
        LET USATotal = USATotal + PopulationFigure_
            (StateSub%, CountySub%)
    NEXT CountySub%
NEXT StateSub%
PRINT "Total USA Population is "; USATotal
END
```

为装入上述带 500 个(五十个州，每个州 10 个郡)人口数字的数组，可编程如下:

```
OPTION BASE 1
DIM PopulationFigures(50, 10)
FOR StateNo% = 1 TO 50
    FOR CountyNo% = 1 TO 10
        PRINT "Enter Population For State "; StateNo%; _
              "County "; CountyNo%
        INPUT PopulationFigures (StateNo%, CountyNo%)
    NEXT CountyNo%
NEXT StateNo%
END
```

或者，采用另一种替代方法，用 READ 和 DATA 语句按如下形式读入该数组:

```
OPTION BASE 1
DIM PopulationFigures (50, 10)
FOR StateNo% = 1 TO 50
    FOR CountyNo% = 1 TO 10
        READ PopulationFigures (StateNo%, CountyNo%)
    NEXT CountyNo%
NEXT StateNo%
'*************************************
'* There would be 50 DATA statements *
'* one per State                     *
'* Each DATA statement would contain *
'* 10 County populations             *
'*************************************
DATA ...              '10 Figures per line
  :        { 50 DATA lines
  :
END
```

如果要运行前面使用二维数组的程序，用户将发现要花费几分钟时间，因为所有的元素都要处理。回忆一下，前面讨论的 Temp 数值，其元素竟达 8760 个!

上面各例中，可用 DO WHILE 或 DOUNTIL 取代 FOR … NEXT 语句，但是 FOR … NEXT 更适合，因为:

(1) 对下标自动增加;

(2) 它指示出循环执行的准确次数。

下标多于 1 个的数组统称为多维数组，各下标用逗号分开。要记住，第一个下标是主项或称行，第二个下标是付项或称列。

自我测验(六)

使用上面定义的 Populationfigures 数组

1. 写出打印 50 个州人口数字的程序。
2. 编程打印人口最多的州的编号
3. 写出人口最多的州的~~编号及人口~~最多的郡的编号程序。
4. 写出打印人口超过1,000,000的郡的个数程序。
5. 写出打印人口超过10,000,000州的个数程序。

§ 10.8.2 二维数组用于累加总和

建立某公司 10 个推销员一年 12 个月销售量的数组。每当有销售时，就把销售量累加到该推销员相应的月份上。

这些数据存储到如下的维数组中:

```
OPTION BASE 1
DIM SalesTotals(10, 12)
```

数组初始化如下:

```
ERASE SalesTotals
```

or

```
FOR I% = 1 TO 10
    FOR J% = 1 TO 12
        LET SalesTotals (I%, J%) = 0
    NEXT J%
NEXT I%
```

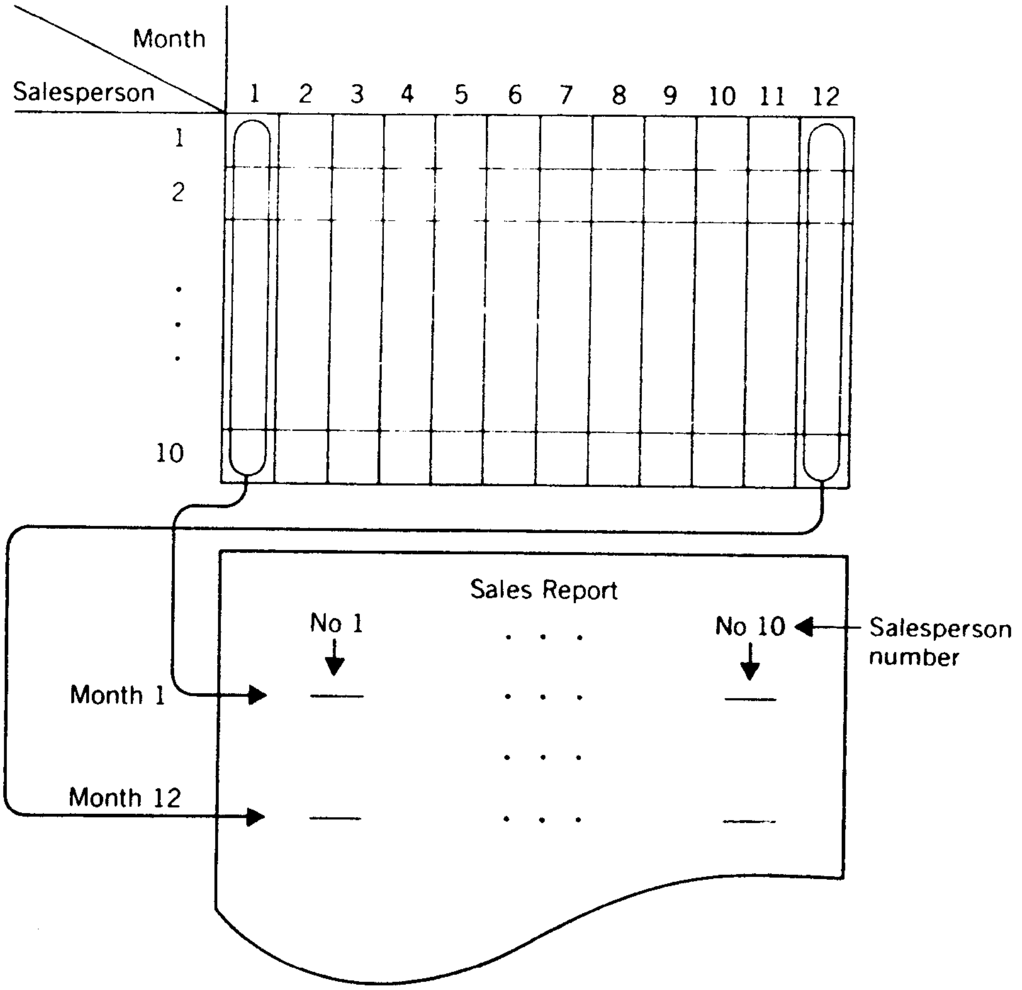
下面程序把输入数据累加到数组中:

```
'program enters sales no, month and amt
'program prints out 12 lines of monthly totals
'each line has 10 sales totals
CLS
OPTION BASE 1
DIM SalesTotals (10, 12)
ERASE SalesTotals
```

```
More$ = "Y"
DO WHILE More$ = "Y" OR More$ = "y"
        PRINT "Enter Sales No"
        INPUT SalesNo%
        PRINT "Enter Month"
        INPUT MonthNo%
        PRINT "Enter Amt"
        INPUT Amt
        IF SalesNo% > 0 AND SalesNo% < 11 AND _
           MonthNo% > 0 AND MonthNo% < 13 THEN
                SalesTotals (SalesNo%, MonthNo%) =_
                    SalesTotals (SalesNo%, MonthNo%) + Amt
        ELSE
           PRINT "Re-Enter Data"
        END IF
        PRINT "MORE INPUT(Y/N?)"
        INPUT More$
LOOP
'Print procedure goes here
```

使用嵌套式 FOR … NEXT 循环，可按行打印出 10 个推销员推销总和，每月一行可打出 12 行数据。

Company Sales Array



我们所做的只是读入这 10 行 12 列的数组然后按照顺序打印出来，这个顺序不同于它的存储顺序。

完整的程序如下：

```
'program enters sales no, month and amt
'program prints out 12 lines of monthly totals
'each line has 10 sales totals
CLS
OPTION BASE 1
DIM SalesTotals (10, 12)
ERASE SalesTotals
More$ = "Y"
DO WHILE More$ = "Y" OR More$ = "y"
        PRINT "Enter Sales No"
        INPUT SalesNo%
        PRINT "Enter Month"
        INPUT MonthNo%
        PRINT "Enter Amt"
        INPUT Amt
```

```
    IF SalesNo% > 0 AND SalesNo% < 11 AND _
       MonthNo% > 0 AND MonthNo% < 13 THEN
            SalesTotals (SalesNo%, MonthNo%) =_
                 SalesTotals (SalesNo%, MonthNo%) + Amt
           ELSE
              PRINT "Re-Enter Data"
           END IF
           PRINT "MORE INPUT(Y/N?)"
           INPUT More$
   LOOP
   LPRINT CHR$(12)
   LPRINT TAB(30); " SALES REPORT"
   LPRINT
   LPRINT "        ";
   FOR SalesCtr% = 1 TO 10
           LPRINT  " No ";
           LPRINT USING "**"; SalesCtr%;
   NEXT SalesCtr%
   LPRINT
   LPRINT
   FOR MonthCtr% = 1 TO 12
           LPRINT "MONTH";
           LPRINT USING "**"; MonthCtr%;
                   LPRINT " ";
                   FOR SalesCtr% = 1 TO 10
                          LPRINT USING "**.**"; _
                               SalesTotals(SalesCtr%, MonthCtr%);
                          LPRINT " ";
                   NEXT SalesCtr%
           LPRINT
   NEXT MonthCtr%
   END
```

输入、输出样例如下:

输入样例

```
Enter Sales No
? 1
Enter Month
? 2
Enter Amt
? 20
MORE INPUT(Y/N?)
? Y
Enter Sales No
? 3
Enter Month
? 6
Enter Amt
? 15
MORE INPUT(Y/N?)
? Y
Enter Sales No
? 1
Enter Month
? 2
Enter Amt
? 12
```

```
MORE INPUT(Y/N?)
? Y
Enter Sales No
? 4
Enter Month
? 10
Enter Amt
? 14
MORE INPUT(Y/N?)
? Y
Enter Sales No
? 3
Enter Month
? 5
Enter Amt
? 15
MORE INPUT(Y/N?)
? Y
Enter Sales No
? 3
Enter Month
? 8
Enter Amt
? 30
MORE INPUT(/N?)
? N
```

输出样例

SALES REPORT

	No 1	No 2	No 3	No 4	No 5	No 6	Nc 7	No 8	No 9	No 10
MONTH 1	0.00	0.00	0.00	0.00	0.00	0.00	0.0C	0.00	0.00	0.00
MONTH 2	32.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 5	0.00	0.00	15.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 6	0.00	0.00	15.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 8	0.00	0.00	30.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH 9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH10	0.00	0.00	0.00	14.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MONTH12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

自我测验(七)

一个数组里存放 30 个班的分类数字即各班专业编号还存放各班注册的男生及女生数.

1. 写出该数组定义语句
2. 对该数组初始化
3. 先写出为输入30个班级专业编号的程序，然后输入每个学生的数据其中包括学生的专业编号及性别（1＝男性，2＝女性），最后统计出各班男生及女生数.

4. 写出计算打印各班注册学生数目的程序。

5. 编写程序打印注册人数超过35名的班级专业编号。

本章我们已讨论了一维和二维数组；一维数组的元素仅一列。此外，还用到二维并行阵列，并将它用作表格；二维数组是行列矩阵。下一节将介绍单个二维数组也可用作表格。

注意：数组维数的增加将大幅度地增大存储空间；例如：Array（20，10，5）有20×10×5＝1000个元素。处理这类数组要花费好长时间。在 Turbo Basic 程序中，维数最多为 8 维。每维的最多元素数为 32，768；一般来说，对于大多数使用者，这一范围已足够。

§10.8.3　二维数组用于存储表格

我们再来考虑存储五位邮政编码及相应税率的问题。在前一节中，用两个单独的表格：一个存储邮政编码，另一个存储相应的税率。

存储该表格的第二种方法是用带有行、列的二维数组：

	Zip	Tax
1	(1，1)	(1，2)
⋮	⋮	⋮
15	(15，1)	(15，2)

该表格维数用 DIM Taxtable（15，12）定义，并在前面加入：

```
OPTION  BASIE  1
```

OPTION BASE 1 可确保 TaxTable 的下标从（1，1）到（15，2）变化，而不是从（0，0）到（14，1）变化。括号内第一个下标是行下标，第二上下标是列下标。

付标或第二下标是 1 的元素代表邮政编码，即表格变元。付标是 2 的元素代表税率，即表格函数。

如果有交易，即当 Zip＝TaxTable（I，1），那么相应于邮政编码的税率就应当为：taxTable（I，2）。

图 10.5 所示为读入税表和客户数据，以及用二维数组计算总和的完整程序：

```
CLS
OPTION BASE 1
DIM TaxTable (15,2)
GOSUB LoadTable
GOSUB GetData
DO WHILE Zip <> 0
        GOSUB FindTaxRate
        GOSUB GetData
LOOP
PRINT "End of Run"
END
LoadTable:
        FOR I% = 1 to 15
                PRINT "Enter Zip Code"
                INPUT TaxTable (I%,1)
                PRINT "Enter Tax Rate"
                INPUT TaxTable (I%,2)
        NEXT I%
        PRINT "Table is full"
RETURN
GetData:
        PRINT "Enter zip code (0 when done)"
        INPUT Zip
        PRINT "Enter unit price"
        INPUT UnitPrice
        PRINT "Enter quantity"
        INPUT Qty
RETURN

FindTaxRate:
        LET I% = 1
        DO WHILE I% < 16 AND Zip <> TaxTable (I%,1)
                I% = I% + 1
        LOOP
        IF I% > 15 THEN
                PRINT "No match found for zip code"
        ELSE
                Total = (1 + TaxTable (I%, 2)) * UnitPrice * Qty
                PRINT "Total due is "; Total
        END IF
RETURN
```

图 10.5 建立使用二维数组程序

例:

在计算机里以下面格式存储一表格:

Table Entry	Maximum Salary	Federal Tax Rate
[1]	10000	.10
[2]	25000	.12
[3]	40000	.20
[4]	55000	.22
[5]	100000	.26

注: 1. 这些税率仅为示例而用，并不代表实际税率。
2. 假定每位雇员工资小于100000。

3. 如果一个雇员工资不到10000，那么其联帮税率为10%；如果雇员的工资在10000–25000之间，则联帮税率为12%，等等.

写出读入该表格的程序，然后输入每位雇员的名字、工资，并打印出雇员的名字、工资、联帮税收、净收入.

净收入 = 工资–税收.

税　收 = 工资×相应的税率.

程序如图10.6所示:

```
'***************************************************
' This program stores tax information in a two-
' dimensional table that is used for computing
' each employee's net pay
'***************************************************
GOSUB Initialize
GOSUB LoadTable
GOSUB EnterInput
DO WHILE Salary < 100000
        LET I% = 1
        GOSUB CalculateTax
        GOSUB PrintResults
        GOSUB EnterInput
LOOP
END
'*******************************************************
' This routine initializes areas in storage necessary
' for processing
'*******************************************************
Initialize:
        CLS
        OPTION BASE 1
        DIM TaxTable(5, 2)
        ERASE TaxTable
RETURN
'*******************************************************
' This routine loads the table using READ and DATA
' statements
'*******************************************************
LoadTable:
        FOR I% = 1 TO 5
                FOR J% = 1 TO 2
                        READ TaxTable(I%, J%)
                NEXT J%
        NEXT I%
DATA 10000,.10,25000,.12,40000,.20,55000,.22,100000,.26
RETURN
'*******************************************************
' This routine prompts for and enters input
'*******************************************************
EnterInput:
        PRINT "Enter Employee Name:";
        INPUT EmpName$
        PRINT "Enter Salary (100000 or more when done):"
        INPUT Salary
RETURN
'*******************************************************
' This routine looks up the tax rate from the table and
' computes net pay
'*******************************************************
CalculateTax:
        I% = 1
        DO WHILE Salary >= TaxTable(I%, 1)
                I% = I% + 1
```

```
        LOOP
        FedlTax = Salary * TaxTable(I%, 2)
        NetPay = Salary - FedlTax
RETURN
'**********************************************************
' This routine prints the results
'**********************************************************
PrintResults:
        LPRINT "Employee Name: "; EmpName$
        LPRINT "Salary: "; Salary
        LPRINT "FedlTax: ";
        LPRINT USING "##,###.##"; FedlTax
        LPRINT "Net Pay: ";
        LPRINT USING "##,###.##"; NetPay
        LPRINT
        LPRINT
RETURN
```

图 10.6(a) 工资单程序清样

```
Enter Employee Name:? Paul Newman
Enter Salary (100000 or more when done):
? 8000
Enter Employee Name:? Robert Redford
Enter Salary (100000 or more when done):
? 60000
Enter Employee Name:? Linda Evans
Enter Salary (100000 or more when done):
? 30000
Enter Employee Name:? Nancy Stern
Enter Salary (100000 or more when done):
? 20000
Enter Employee Name:? Stop
Enter Salary (100000 or more when done):
? 999999
```

图 10.6(b) 输入样例

```
Employee Name: Paul Newman
Salary:  8000
FedlTax:    800.00
Net Pay:  7,200.00


Employee Name: Robert Redford
Salary:  60000
FedlTax: 15,600.00
Net Pay: 44,400.00


Employee Name: Linda Evans
Salary:  30000
FedlTax:  6,000.00
Net Pay: 24,000.00


Employee Name: Nancy Stern
Salary:  20000
FedlTax:  2,400.00
Net Pay: 17,600.00
```

图 10.6(c) 输出样例

§10.9 折半查询二维数组

§10.9.1 串行查询

到现在为止，我们一直讨论的查询方法就是所谓串行查询（serialsearch）

> **串行查询**
>
> 1. 先查询表格的第一数据
> 2. 如果条件满足，则查表结束.
> 3. 如果条件不满足，下标加1，再查表格的第二个数据
> 4. 这个过程一直进行到找到的匹配值或表格全部查询完毕.

对下面两种情况，最好使用这里所讲的顺序查表或叫串行查表法.

1. 表格内的输入数据，既不是升序排列，也非降序排列，即它们是随机存储的.
2. 表格数据经过处理. 通常，头几个值就是要找的值. 这样，可以缩短存取时间；因为头几次比较判断后就可适当地终止查表.

然而，在许多例子中，表格数据是按某数字顺序排列的；例如，在 Dis Count Table 数组中，50 个客户的序号在表格内按升序排列.

DiscountTable

	TCustomerNo	TDiscountPct
TCustomerNo is in ascending sequence	0100	2.0
	0200	1.0
	0400	5.0
	⋮	⋮

表该格包含有每个客户所享有的百分比贴现率. 注意，尽管客户的序号是顺序排列的，但不一定是接连的.

可如下定义该数组表格:

```
OPTION  BASE 1
DiscountTable (50, 2)
```

在有些表格数据按序排列的情况下，例如在上例中，串行查表效率就不高. 若要查找序号为 9000 号客户的 TDiscountPct，从表头开始找是很费时的. 因为表格是有序排列

的，那么第 9000 号客户大概在表格的尾部；因此，从表头开始是浪费时间的。

§ 10.9.2 折半查询

当表格按某些表格输入项或表格变元有序排列时，折半查询（Dinary Ceareh）是效率最高的查表法。

下面是折半查表的步骤:

1. 开始把查询变元同表格的中间变元比较一下；例如，如果有50个表格变元，那么就把查询变元同第 25 个表格变元比较。如果有 27 个表格变元，就把查询变元同第 14 个（四舍五入）表格变元比较；也就是说，开始把输入记录的 CustomerNo 同表格中间的 TcustomerNO 比较。
2. 如果查询变元大于表格中间变元，则省去对表格第一半的查询。在所举的示例中，表格上有 50 个输入条目；假定查询变元 CustomerNO 是 9000，它大于第 25 号表格变元，则在搜索过程中就可省掉对上半部的查询。
3. 另一方面，如果查询变元小于表格的中间变元，就省略对表格第二半的检查；接着，再把表格的第一半一分为二，把查询变元同第 13（25／2 四舍五入）个表格变元比较。
4. 这样连续地把表格一劈二半，直到找到匹配值为止。

平均来讲，折半查询找匹配值比串行查询比较次数要少得多。

例如，假定 CustomerNO ＝ 5000:

Discount Table

	Table entry	TCustomerNo (Table argument)	TDiscountPct (Table function)
	1.	0100	2.0
	2.	0200	1.0
	3.	0400	5.0
	4.	0500	3.1
	⋮		
1. 1st compare	25.	4300	4.3 (>)
	⋮		
3. 3rd compare	31.	4890	8.4 (>)
	⋮		
4. 4th compare	34.	5000	5.6 (=) *
	⋮		
2. 2nd compare	37.	5310	2.4 (<)
	⋮		
	50.	9940	7.1

当同表格的第 34 项比较时，就可找到 CustomerNO 的匹配值 tcustomerNO. 如果用串行查表，则需要进行 34 次比较；而使用交互式折半查表法，则只需要进行 4 次比较.

这种交互查询方法就称做折半查询，因为每次比较都省去所剩数据项的一半不用管. 也即，每次比较都把要考虑的因素减少一半.

下面的情况使用折半查询比串行查询要好：

> **折半查询的用途**
>
> 1. 当表格数据项按某种顺序——升序或降序排列时.
> 2. 当要查询或检索的有序表格的数据项比较大（如50或更多）时.

对小表或那些数据项不按次序排列的表格，表查询可用标准的串行查表法；对大表格，其数据项按特定的顺序排列，折半查询是效率最高的. 所谓大表格要说清楚有一定困难，但是对于按某一顺序排列的超过 50 个条目的表格而言，使用折半查询法可能是有益处的.

§ 10.9.3 折半查询的示例

先读入含 25 个元素的 customerTable 表，按 TCustomerNO 顺序排列，其中包含有每位客户的贴现率；然后再读入一记录文件，求出每位客户所享有的贴现量. 交易包括 CustomerON 和 AmountOfSale，打印书 Netprice＝Amount Of Sale－Discount.

使用折半查询从表中找出贴现率百分比(图 10.7 为答案).

```
'**********************************************
'This program reads in 25 customer numbers and
'their discount percents and stores the data
'in a table.  Then each customer's amount of
'sale is entered as input and the net balance
'due is calculated with a binary search of the
'table
'**********************************************
CLS
OPTION BASE 1
DIM CustomerTable(25, 2)
PRINT "Enter Table First"
FOR I% = 1 TO 25
        J% = 1
        PRINT "Enter Customer Number"
        INPUT CustomerTable(I%, J%)
        J% = 2
        PRINT "Enter Discount Percent"
        INPUT CustomerTable(I%, J%)
NEXT I%
PRINT "Table is Filled"
PRINT
PRINT "Enter Customer Number;"
INPUT CustomerNo
PRINT "Enter Amount of Sale;"
```

```
INPUT AmountOfSale
DO WHILE CustomerNo < 99999
'*******************************************
'The following is a binary search with lower
' = 1 and upper = 25; in reality, any two
' values could be used
'*******************************************
        Lower = 1
        Upper = 25
        DO WHILE Upper >= Lower
                Sub1 = (Lower + Upper) \ 2
                IF CustomerNo = CustomerTable(Sub1, 1) THEN
                        Upper = 0
                        Lower = 1
                ELSEIF CustomerNo < CustomerTable(Sub1, 1) THEN
                                        Upper = Sub1 - 1
                ELSE
                        Lower = Sub1 + 1
                END IF
        LOOP
        IF CustomerNo = CustomerTable(Sub1, 1) THEN
                NetPrice = AmountOfSale - AmountOfSale * CustomerTable(Sub1)
                PRINT "Net Price is "; NetPrice
        ELSE
                PRINT "Customer Number not found"
        END IF
        PRINT "Enter Customer No:"
        INPUT CustomerNo
        PRINT "Enter Amount of Sale"
        INPUT AmountOfSale
LOOP
END
```

图 10.7 使用折半查询法的程序

§ 10.10 内部数组与外部数组

前面，我们举例说明了作为表格的数组既可输入数据赋值，也可用 READ 和 DATA 语句读数赋值；把数据输入表格数组可以使用键盘，也可从磁盘文件读入；无论哪种情况，它们统称为"外部表格"，因为它们是在交易记录处理之前读进来的。如果表格存储在一个文件中，当表格有新数据需要修改时，则还需一个单独的程序。

注意到，有些表格不是输入赋值，而是使用 READ … DATA 语句编程赋给常数值；这样的表格叫"内部表格"。当数据似乎不需改变时，就应该把它们存储到内部表格中。如果必须修改内部表格的值，就得对表格查询程序本身进行修改，这往往是很危险的；因此，只有对表格元素变动极少时才编制内部表格程序。因为任何改变都可能导致错误，所以程序员一般不愿对运行程序做修改。内部表格比外部表格使用频度要小，因为通常对输入数据项还是有可能改动的。

§ 10.11 本章辅导

§ 10.11.1 本章小结

一、数组介绍

1. 数组用于存储同类型的一串数据元素
2. 数组用于累计总和
3. 数组可存储用于数据“检索”的表格

二、用数组存储数据元素

1. DIM语句定义数组的大小

① 如果数组元素不多于11个，不用DIM语句，

② DIM语句必须放置在使用数组的前面

③ 数组可存储数字量或字符串数据

2. 使用下标访问数组元素

① 下标：用园括号括起来，用来指出要用数组中哪个元素

② 在没有其它任何说明时，元素的第一个元素默认元素.

③ 要指明一数组的第一个元素是元素1，可用下列代码：

```
OPTION BASE   1
```

④ 下标可以是整数、变量或算术表达式.

3. 用FOR … NEXT处理数组.

① FOR … NEXT可用来对下标初始化及使下标增加

② 也可用其它循环结构，但需要较多的语句.

4. 如果要读入数据，数组可不必初始化

三、用数组储存总和

1. 数组的元素可以是输入数据的累加总和
2. 数组用来累加数字时，处理之前必须要初始化
3. ERASE语句可对数组初始化

① 对数字元素初始化为零

② 对字符串数组元素则初始化为空

③ FOR … NEXT循环可取代REASE，分别对数组的每个元素初始化.

4. 在输入数据累加到数组元素之前，最好对其进行有效化处理.
5. 所有数据均处理完毕后，一般地应把数组内容打印出来.

四、数组处理的几个补充概念.

1. Turbo Asic的Bounds可选项应该设置断ON，以确保下标是有效的.

① 如果不置Bounds ON，则不做下标越界错误的检查.

② 如果Bounds可选项是OFF，超出下标范围的数据将复盖内存中其它的输

入数据.

2. 动态数组和静态数组。

① STATIC数组大小固定

② 除非特别说明，否则所有数组都默认为STATIC数组

③ DYNAMIC数组大小可变

a. 在其处理前的运行过程中输入其大小.

b. REASE语句删除所有的DYNAMIC数组.

④ $ DYNAMIC和 $ STATIC叫做元语句.

a. $ DYNAMIC及 $ STATIC元语句后面定义的任何数组，分别被认为是动态或静态数组.

b. 因为缺省即为静态数组，故 $ STATIC元语句可省.

五、用数组存储表格数据

1. 表格就是一列数据项，当处理过程中需要特定的条目时把它作为参照.

2. 两个并行数组可用来"查表".

① 一个数组存放表格变元，它要同读入的交易数据比较.

② 另一个数组存放表格函数，函数值即为要查询的值.

③ 并行数组的元素数目相同.

3. 数据存入表格可减少输入操作，使得更新数据的处理效率高.

4. 要找到一个条目，得把数组数据同交易数据比较来查表.

① 在处理交易数据之前，必须读入表格数据.

② 用来查匹配值的交易数据就是查询变元.

③ 在匹配条目找不到时，应该编出错程序.

5. 当表格变元既是顺序的，又是连续的时，就不必放入表格中；也就是说，在该条目位置上可直接读取表格函数.

六、多维数组.

1. 多维数组有几维就需几个下标

① Turbo Basic最多允许 8维.

② 数组的大小限制在64K的存储空间内.

2. 多维数组可用来累加总和或存储表格.

七、内部及外部表格.

1. 在交易数据处理之前，读入外部表格.

2. 程序自身内部的一串常数编成内部表格，通常用READ、DATA语句读入数据.

① 改变内部表格的内容需要对表格查询程序修改.

② 内部表格不及外部表格使用频繁.

§ 10.11.2 本章自测

1. (TorF) 下标可以是常数或作为输入读入的变量.
2. 看下面的程序

```
OPTION BASE1
DIM Instructor Name$ (15)
```

上面数组由一个字符串变量组成，写出建立这一数组的另外一种语句的方法.
3. 某院校有5个班，每班肯8名学生，每位学生都参加了一门期终考试. 写出存储期终考试成绩数组的 OPTION 和 DIM 语句.
4. 需输入的考试成绩数为　　　个.
5. 写出把全部考试成绩读入数组的程序.
6. 写出打印各班平均分数的程序.
7. 写出打印各班成绩在90分以上的学生数.
8. 写出打印全校范围平均成绩的程序.
9. 写出打印各班最好成绩的程序.
10. 写出打印成绩达到100分的班级数.

§ 10.11.3 实例

一个商店分三个部门，要把各部门星期一到星期六的日销售量通过键盘输入一外部数组；数组装完后，要计算总和，并打印下列内容:

1. 各部门一周的总销售量.
2. 商店每天的总销售量.
3. 商店一周的总销售量.

由于没有指出按某一顺序输入数据，因此需先输入部门及日期，然后再放到适当的数组元素中(参见图 10.8 中的层次图、程序及输入、输出样例). 注意，我们把部门的总销售量存入 SalesAmt (Dept%, 7) 中，日总销售量存入 SalesAmt (4, Day of week%)，因此数组说明为 (4, 7)，而不是 (3, 6).

```
START
     Dimension the array
     PERFORM
          Clear the screen
          Clear the array
          Set MoreData$ to "Y"
     ENDPERFORM
     DO WHILE MoreData$ = "Y"
          PERFORM
               Enter input
               Add to array element
          ENDPERFORM
     ENDDO
     PERFORM
          Calculate Totals
     ENDPERFORM
     PERFORM
          Print Dept Totals
     ENDPERFORM
     PERFORM
          Print Daily Totals
     ENDPERFORM
     Print Final Total
STOP
```

图 10.8(a) 伪代码

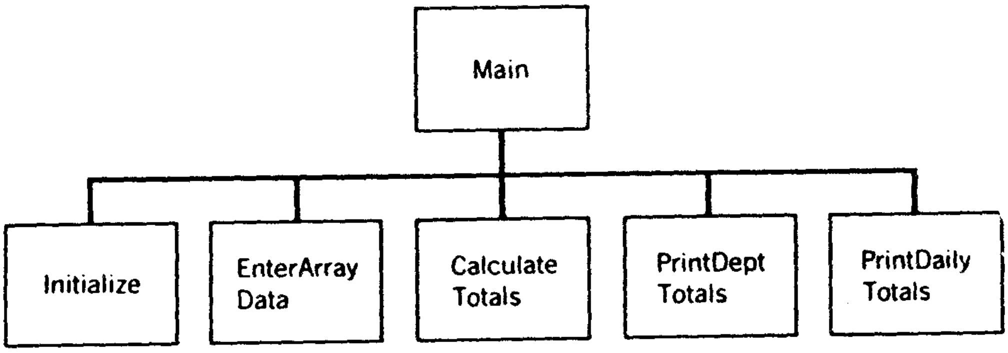


图 10.8(b) 层次图

```
******************************************
 This program demonstrates the use of a
 two dimensional array for accumulating
 totals
******************************************
GOSUB Initialize
DO WHILE MoreData$ = "Y"
        GOSUB EnterArrayData
LOOP
GOSUB CalculateTotals
GOSUB PrintDeptTotals
GOSUB PrintDailyTotals
LPRINT "Total sales for the week: ";
LPRINT USING "$$##,###.##"; SalesAmt(4, 7)
LPRINT
LPRINT "End of Report"
END
```

```
'*********************************************
'This routine adds data to the array
'*********************************************
EnterArrayData:
        PRINT "Enter Dept No (1 - 3): ";
        INPUT Dept%
        PRINT "Enter Day of Week ( 1 - 6): ";
        INPUT Day%
        PRINT "Enter Sales for that Dept on that Day: ";
        INPUT Amt
        IF Dept% >= 1 AND Dept% <= 3 AND Day% >= 1 AND Day% <= 6 THEN
                SalesAmt(Dept%, Day%) = SalesAmt(Dept%, Day%) + Amt
        ELSE
                PRINT "Data Error - Please Re-enter"
        END IF
        PRINT "More Data (Y/N)?"
        INPUT MoreData$
RETURN
'*********************************************
' This routine calculates the necessary totals
'*********************************************
CalculateTotals:
    FOR Dept% = 1 TO 3
        Total = 0
        FOR DayOfWeek% = 1 TO 6
                Total = Total + SalesAmt(Dept%, DayOfWeek%)
        NEXT DayOfWeek%
        SalesAmt(Dept%, 7) = Total
    NEXT Dept%
    FOR DayOfWeek% = 1 TO 7
        Total = 0
        FOR Dept% = 1 TO 3
                Total = Total + SalesAmt(Dept%, DayOfWeek%)
        NEXT Dept%
        SalesAmt(4, DayOfWeek%) = Total
    NEXT DayOfWeek%
RETURN
'*********************************************
' This routine prints the dept totals
'*********************************************
PrintDeptTotals:
        LPRINT TAB(15); "Department";
        LPRINT TAB(30); "Weekly Sales"
        LPRINT
        FOR Dept% = 1 TO 3
                LPRINT TAB(20); Dept%;
                LPRINT TAB(30);
                LPRINT USING "$$#,###.##"; SalesAmt(Dept%, 7)
        NEXT Dept%
RETURN
'*************************************************
' This routine prints the daily totals
'*************************************************
PrintDailyTotals:
        LPRINT
        LPRINT
        LPRINT
        LPRINT TAB(15); "Day";
        LPRINT TAB(30); "Total Sales"
        LPRINT
        FOR DayOfWeek% = 1 TO 6
                LPRINT TAB(15); DayOfWeek%;
                LPRINT TAB(30);
                LPRINT USING "$$#,###.##"; SalesAmt(4, DayOfWeek%)
        NEXT DayOfWeek%
        LPRINT
        LPRINT
RETURN
'*************************************************
' This routine initializes variables and the array
'*************************************************
```

```
Initialize:
        CLS
        OPTION BASE 1
        DIM SalesAmt(4, 7)
        ERASE SalesAmt
        MoreData$ = "Y"
RETURN
```

图 10.8(c)　程序

```
Enter Dept No (1 - 3): ? 2
Enter Day of Week ( 1 - 6): ? 5
Enter Sales for that Dept on that Day: ? 50.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 1
Enter Day of Week ( 1 - 6): ? 4
Enter Sales for that Dept on that Day: ? 30.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 1
Enter Day of Week ( 1 - 6): ? 4
Enter Sales for that Dept on that Day: ? 20.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 3
Enter Day of Week ( 1 - 6): ? 2
Enter Sales for that Dept on that Day: ? 300.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 2
Enter Day of Week ( 1 - 6): ? 1
Enter Sales for that Dept on that Day: ? 50.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 3
Enter Day of Week ( 1 - 6): ? 6
Enter Sales for that Dept on that Day: ? 40.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 1
Enter Day of Week ( 1 - 6): ? 2
Enter Sales for that Dept on that Day: ? 45.00
More Data (Y/N)?
? Y
Enter Dept No (1 - 3): ? 3
Enter Day of Week ( 1 - 6): ? 5
Enter Sales for that Dept on that Day: ? 60.00
More Data (Y/N)?
? N
```

图 10.8(d)　输入样例

```
Department        Weekly Sales

     1               $95.00
     2              $100.00
     3              $400.00


Day               Total Sales
```

```
                    1                      $50.00
                    2                     $345.00
                    3                       $0.00
                    4                      $50.00
                    5                     $110.00
                    6.                     $40.00

      Total sales for the week:       $595.00

      End of Report
```

图 10.8(e)　输出样例

§ 10.11.4　复习题:

1. 下面的程序打印出什么结果.

```
OPTION BASE 1
DIM Array1 (2, 4)
FOR I = 1 TO 2
    FOR J = 1 TO 4
        READ Array1 (I, J)
    NEXT J
    RESTORE
NEXT I
DATA 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23
PRINT Array1 (1, 3), Array1 (2, 1), Array1 (1, 4)
```

2. 如果省略RESTORE语句, 上面程序打出什么结果.
3. 写出把目录表以三个并行数组形式读入的程序: 第一个数组存放20种零件编号, 第二个存放各种零件的单价, 第三个数组存放各种零件的库存量.
4. 写出读入零件编号的程序段, 并用查表方法找出该零件的单价及库存量.
5. 使用折半查询法重写问题4的程序.
6. 上面问题在什么情况下可用折半查询?

§ 10.11.5　调试练习

1. 下面程序错在哪儿? 找出并改正错误.

```
DIM TransTable (1:200)
ERASE TransTable
PRINT "Enter Number of Transactions: ";
INPUT TransCount
FOR Count = 1 TO TransCount
    PRINT "Enter Transaction Amount"
    INPUT TransAmt (TransCount)
NEXT Count
```

2. 下面程序为什么输出错误的结果？改正其错误.

```
CLS
DIM SalesTotal (1:20)
DO  WHILE SalesNo <> 0
    PRINT "Enter Salesperson number, sales amount"
    INPUT SalesNo, SalesAmt
    SalesTotal (SalesNo) = SalesTotal (SalesNo) + SalesAmt
LOOP
PRINT "Sales Person", "Total Sales"
FOR I% = 1 TO 20
    PRINT I%, SalesTotal (I%)
END
```

3. 假定问题2推销员的编号既不连续也无顺序，修改程序使适合推销员编号为非连续的情况.

识别并更正下面程序段的错误:

4.
```
OPTION BASE 1
DIM Item (3, 5)
FOR ItemNo = 1 TO 5
    FOR Price = 1 TO 5
        INPUT Item (ItemNo, Price)
    NEXT Price
NEXT ItemNo
```

5.
```
OPTION BASE 1
DIM Budget (12, 20)
FOR Month = 1 TO 12
    FOR Item = 1 TO 20
        INPUT Budget (Item, Month)
    NEXT Month
NEXT Item
```

§ 10.11.6 编程题:

1. 编写打印各推销人员销售总额的程序. 该题的说明见图10.9.

注:

a. 20个销售人员，编号为1-20.

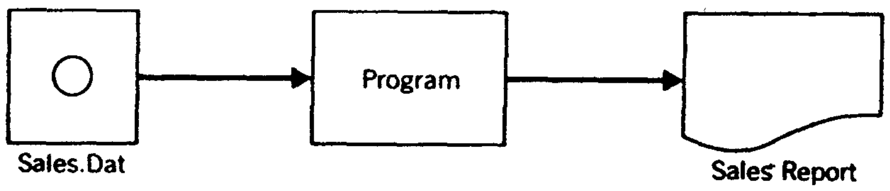
b. 每做成一次交易都用来在磁盘上产生一个输入记录，因此，如果做成的交易多，就会有大量的输入记录.

c. 输入记录不按顺序存放（如果记录按顺序放，可用控制中断过程）.

d. 打印出每位销售人员的总销售数字；虽然输入的记录数是变化的，但输出结果仍是 20 个总和.

e. 应该能对所有总和字段进行编辑.

系统流程图



Sales.Dat 记录格式

SalesPersonNo, SalesPersonName$, AmtOfSale

销售报告打印格式图

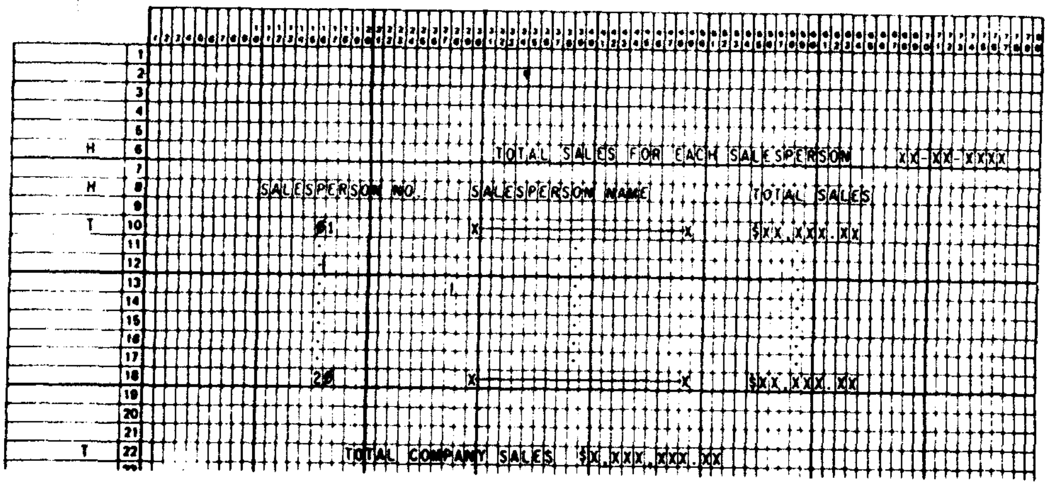


图 10.9 编程题 1 的说明

2. 该问题的定义见图 10.10；其要求如下:

a. 计算ABC公司每位雇员的实际月收入；税率表必须从20个输入磁盘记录中读入内存。例如:

含税工资 (Taxable Income)	联邦税 (Federal Tax)	州税 (State Tax)
09800	0.040	0.010
12000	0.080	0.020

含税工资小于等于9800时，州税为1%，联邦税为4%；含税工资在9801～12000（包括9801-12000）之内时，州税为2%，联邦税为8%；依此类推。

b. 把表格读入并储存好后读入工资文件，月实际收入如下计算：

(1) 标准扣除＝年工资中第一个＄10，000的10%

(2) 从属扣除＝1080×从属数

(3) FICA（社会安全税）＝年工资中第一个＄45，000的7.51%。

(4) 含税工资＝年工资-标准扣除从属扣除。

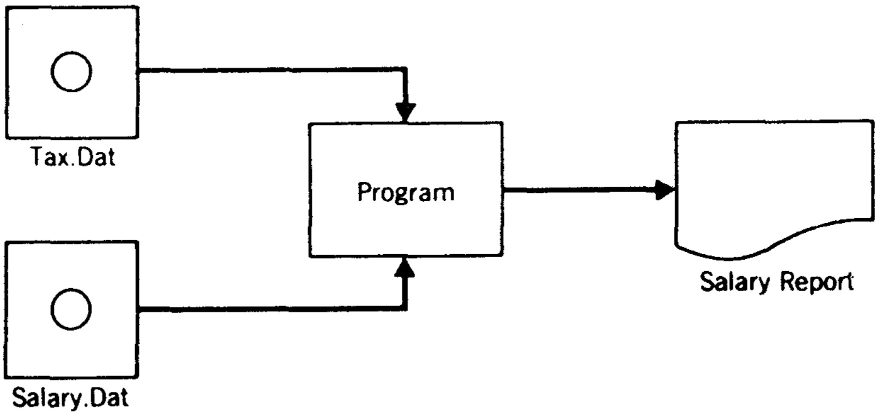
(5) 使用税表求出含税工资的税。

(6) 年实得收入＝年工资-（州税率%×含税工资）-（联邦税率%×含税工资）-FICA

(7) 月实际收入＝年实际收入／12

(8) 打印出每位雇员的名字及相应的月实际收入（经扣除后所得）。

系统流程图



Tax.Dat 记录格式

MaxIncome, FedlTaxRate, StateTaxRate

Salary.Dat 记录格式

EmpNo, EmpName$, AnnSal, NoOfDependents

Bills 工资报表打印格式图

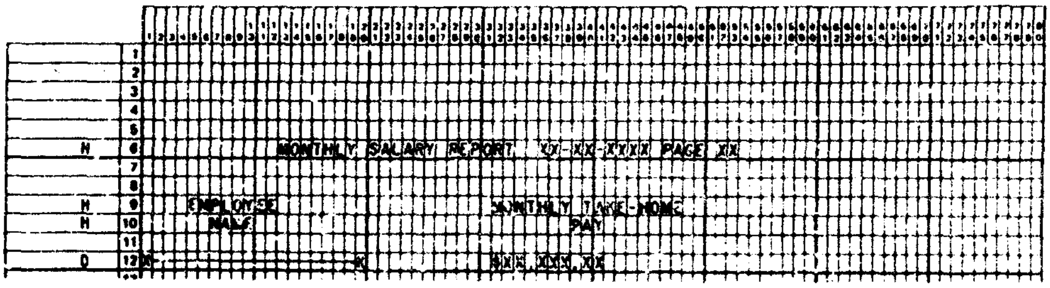


图 10.10　编程题 2 的说明

3. 写出打印销售报告的程序，问题详述见图 10.11。

注：

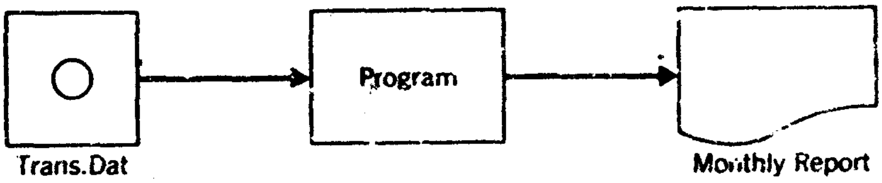
a. 输入记录的数目是可变的，打印六月份每天的日销售总额以及六月份的月销售总额。

b. 输入记录没有顺序。

c. 要对所有数字字段编辑。

d. TransDate中月份从1—12变化。

系统流程图



Trans.Dat 记录格式

CustNo, TransAmt, TransDate (mmddyyyy)

每月报告打印格式图

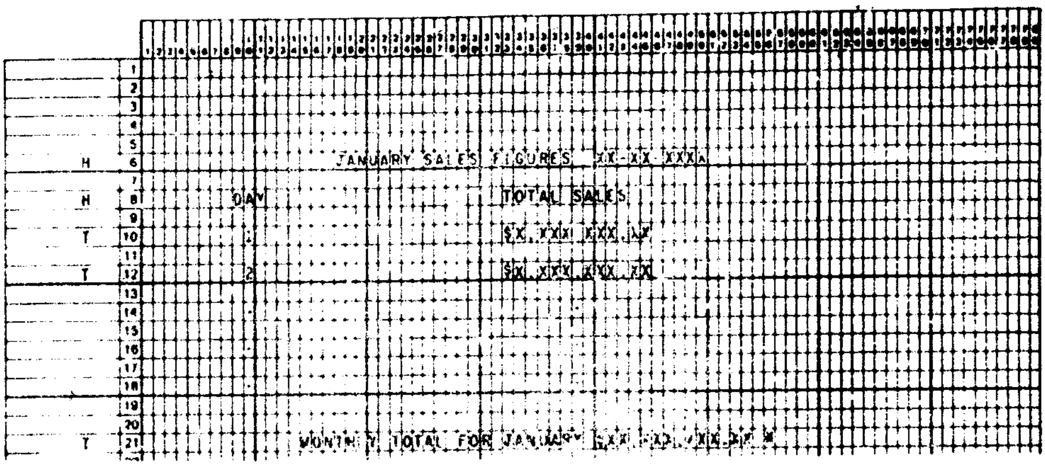


图 10.11　编程题 2 的说明

4. 编程依次打印日销售总额；其数据按下面格式输入：

"Enter Two Amounts Seperated by a Comma"

注:

a. 第1行输入29日和30日的销售量，第2行输入27日、28日的销售量，等等；共有 15 个记录.

b. 按1日－30日顺序打印出销售量；在每输出行上，打印日期及相应的销售量.

c. 输出加上适当的标题和日期.

5. 该题的说明参看图10.12.

注:

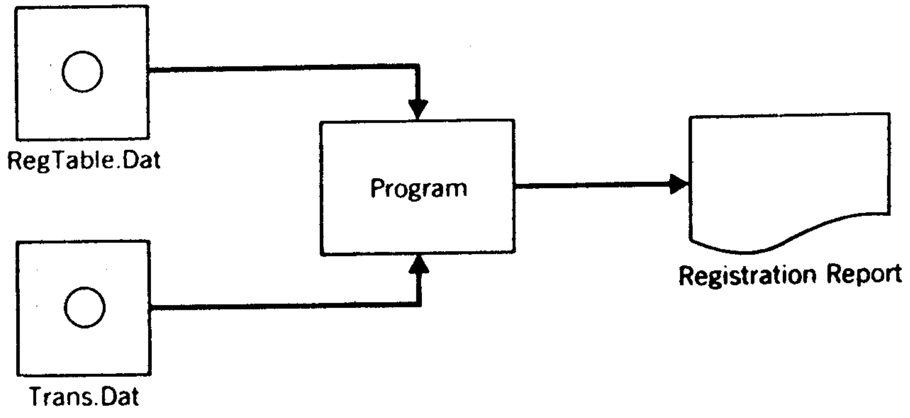
a. 有25个表格输入项.

b. 把表格读入并储存后，读入销售文件.

c. 对每个销售记录，要得到相应的注册等级必须找到机车的类型.

d. 注册费＝机车重量×注册等级（查表可得）.

系统流图



RegTable.Dat 记录格式

VehicleClass, RegistrationRate

Trans.Dat 记录格式

VehicleClass, Name, WeightOfVehicle, VehicleDescription

注册报告打印格式图

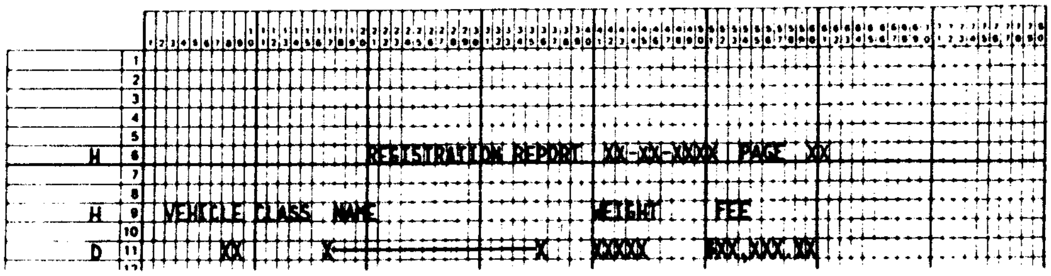


图 10.12　编程题 5 的说明

第十一章 用过程进行自顶向下模块化程序设计详述

本章要求:

1. 掌握用于自顶向下模块化程序之中的设计方法.
2. 熟悉用来规划自顶向下模块化程序的层次图.
3. 掌握自顶向下模块化程序中过程的使用.

§11.1 自顶向下模块化程序设计的回顾

§11.1.1 为什么要编写自顶向下模块化程序

至此，我们已经学习了在编程以前如何使用伪代码来规划和设计程序，而且还学习了如何编写包含设计程序时必须的逻辑控制思想的结构化 Turbo Basic 程序．最后，还学习了如何利用子程序来模块化一个程序．先编写主模块，然后分级编写子模块，紧接着编写更小的模块，以此类推，我们把这种程序设计思想称之为"自顶向下的模块化程序设计"．

要编写一个自顶向下模块化的程序，应象写一份格式已确定的报告那样来书写．也就是说，应当先从一个简单的提纲入手，利用这个提纲来组织报告，只有当报告的总体结构被确定以后，我们才能分头编写各个部分、以及含有段的更详细的章节．类似的，一个程序应该先作规划制定出它的总体设计,这里应当包括分配在程序主体或是主模块中出现的模块的名字，以便于标明这个程序所要完成的功能．只有在主模块里我们已经规划好了整个程序的总体结构，我们才能够编写这些子程序模块．即使是在这些子程序模块中，我们仍旧采取这种方法，即：先考虑子程序模块的总体结构，然后分解成更小的子程序，直至最后落实成为计算机的指令集合．

自顶向下的模块化程序设计有以下几个主要的优点:

§11.1.1.1 一个好的程序应当是自顶向下和模块化相结合的产物

模块化程序很容易编程、调试和修改，因为它们是由一系例段和单元组成的．自顶向下的程序，它的模块编制过程是按模块的重要性程度编写的，即：重要程度高的先编写，次要的后编．因而，这样的程序其总体结构是很突出的．

对于某些非常大而且很复杂的程序，其中的一些模块能被别的程序所调用、或是由另外的程序员所编写的，更有甚者，有些模块可能与主模块或者程序主体是分开单独进行调

试的。因此，我们极力推荐：编制程序时，使用模块来取代完成某一特定功能的、在逻辑上相关联的指令集。以后，对每一个子模块分别进行编码时，再用具体的操作指令去实现。

§ 11.1.1.2 自顶向下的模块化程序减少重复性劳动

假定我们要从程序的不同位置调用执行某一组特定的指令；例如：就象在第六、七章所述的标题例程那样，在程序的开头以及每次遇到页尾时都需要频繁地调用标题例程。如果不使用模块的概念，那么在编程过程中，每次需要这一组指令时，就不得不重复地编上这么一段相同的指令。反之，如果在程序的主模块（或者其它模块）中定义了这样一个模块可供调用，那么在需要它的时候我们就能调用它。

我们已经使用了 GOSUBS 作为子程序来完成一个自顶向下的模块的设计，但是 GOSUB 技术并不是唯一的设计技术。实现模块化还有另外两种技术，即：过程和函数。本章将详细地讨论过程，而函数则放到下一章再讨论。

§ 11.1.2 自顶向下模块化程序中的 GOSUB

程序中的 GOSUB 语句具有如下格式：

```
    :
GOSUB 模块名
    :
END

模块名:
    :
RETUTN
```

我们已经看到，子程序的一个普遍用处是用来打印标题。标题在下列几种情况下是经常遇到的：

① 在一个程序的开头；
② 当某一页到结尾而开始新的一页时（例如：25行的空行）；
③ 报告结尾所在页上需要打印总页数。

此外，标题还可能需要一些数据、页数以及几行的输出结果，而所有这些都会导致许多的编码。最好的方法就是在编主程序时，只是调用能打印标题的子程序，而不是在每一次需要打印标题的地方都编上这么一段相同的程序，如下所示：

```
'Main Module goes here
GOSUB PrintHeadings
     .
     .
     .
DO   WHILE Amt < 999
     IF  LineCounter > 50 THEN
         GOSUB PrintHeadings
     END IF
     .
     .
     .
LOOP
GOSUB PrintHeadings
     .
     .
     .
END
PrintHeadings:
     LPRINT ...
     LPRINT ...
     .
     .
     .
RETURN
```

GOSUB 语句可以放在程序中的任何位置，甚至可以在程序本身是由 GOSUB 语句调用的程序中安排 GOSUB 语句.

由上可以看出，子程序能够通过消除重复指令集的办法来达到简化编码的效果. 在上面的例子中，打印标题的那些指令实际上只进行了一次编码，而不考虑它们真正执行了多少次. 任何时候，当某个程序中要不止一次地执行某一组指令时，就可以把它们编成子模块，然后在需要的地方调用它们. 除此之外，由于对这一组指令只在一处进行了编程，就大大地减少了编程出错的可能性，而且使得程序易于维护. 例如，出于某种原因我们要修改标题例程，则又需要修改一个地方就行了，而不是去修改所有要打印标题的地方.

子程序的编制，也要遵从自顶向下的原则，较抽象、主要的例程先进行编码，具体而次要的后编. 假定打印标题模块被另外两个地方调用，一个是主程序或主模块，另一个模块是启动模块（Getting started）. 而启动模块（Getting started）作为一个更抽象的模块，应当在较具体的模块——打印标题模块——之前编程.

编一个程序时，用户可以先在程序中使用一些模块名用来标明此处所需指令序列的功能，而这些模块却在后来才具体编写.

一个自顶向下模块化程序框架的例子如下:

```
GOSUB InitializeVariables
.
.
.
GOSUB PrintHeadings
.
.
.
GOSUB GetInput
.
.
.
```

```
DO  WHILE Amt < 999
    GOSUB PerformCalculations
    .
    .
    .
    GOSUB GetInput
LOOP
GOSUB PrintTotals
END
'Modules would be coded here
```

当上述的结构安排好以后，再对每一个具体的模块单独进行编程:

```
InitializeVariables:
.
.
.
RETURN
.
.
.
PrintHeadings:
.
.
.
RETURN
.
.
.
```

在每个模块的前面最好加上注解用以描述模块的功能.

作为一个惯例，我们要编写的模块，都应当在主模块编写完以后才开始编写，也就是说，在主模块的 END 之后．重要的子模块先编写，次要的、具体的模块后编.

我们前面所讨论的，每个模块都作为一个子程序来看待．一个模块包含了一组相关操作的指令．在程序的主体中，把一个子程序作为模块来处理要比在主程序中写上一长串指令要好一些，请看下面的例子:

非模块化方法

```
IF  Salary > 100000 THEN
    LET Bonus = 5000
    LET FederalTax = .26 * Salary
    LET StateTax = .1 * Salary
ELSE
    LET Bonus = 2500
    LET FederalTax = .18 * Salary
    LET StateTax = .05 * Salary
END IF
```

```
IF  Salary > 100000 THEN
    GOSUB SixFigureSalaryRoutine
ELSE
    GOSUB StandardSalaryRoutine
END IF
.
.
.
END
SixFigureSalaryRoutine:
    LET Bonus = 5000
    LET FederalTax = .26 * Salary
    LET StateTax = .1 * Salary
RETURN
StandardSalaryRoutine:
    LET Bonus = 2500
    LET FederalTax = .18 * Salary
    LET StateTax = .05 * Salary
RETURN
```

使用模块化方法，那么在主程序中只需要简单地测试一下指定的条件，而把指定条件满足时所要执行的语句先空开，另外编成一个子程序．类似地，指定条件不满足时所要执行的语句也被编成了另一个子程序．用方框框上的指令形成了一个操作集合，这些指令最好编成独立的模块．

这与下面的情况很相似；即，宁可在一页中分成几段来写句子，而不要把所有的句子都堆积在一段里面．用户会发现模块化的程序很容易编写、阅读、调试以及修改．

模块名或者标号都必须以字母打头，后面可以跟任何数目的字母或数字，而且可以用小数点作为分隔符．给模块取名字的时候，最好使用有意义的名字．在一个有许多模块的又大又长的程序中，应给每个模块名前头加上标识码用来指明它在程序中出现的位置．A100，A200，A300，B100，及 B200;等都是较常见的前缀码．举例来说，如果一个模块名前的前缀码为 A200,那么这个模块应当放在前缀码分别为 A100 和 A300 这两个模块之间．

§11.2 在自顶向下模块化程序中使用过程

§11.2.1 基本格式

在这一节里，将介绍过程在自顶向下模块化程序中的作用，它与子程序将有很大的不同．过程是一种比子程序更加模块化的工具,因为它们不仅把执行特定任务的指令给模块化了，而且还把数据处理方法给模块化了．

一个过程是一组相关操作的集合，这组操作集合在主程序中通过 CALL 语句的调用而得以执行，CALL 调用的格式如下：

```
'Main module goes here
 .
 .
 .
CALL procname
 .
 .
 .
END
'***********************************************************
'* All modules--procedures and subroutines--are best coded *
'* after the END                                           *
'***********************************************************
SUB procname
 .
 .
 .
END SUB
```

其中，从 SUB procname 那一行开始，直到 END SUB 为止的语句定义了被 CALL 语句调用的过程。我们可以使用跟 GOSUB 语句同样的方法来使用 CALL 语句：

例:

```
 .
 .
 .
CALL PrintHeadings
 .
 .
 .
DO  WHILE Amt < 999
     .
     .
    IF  LineCtr > 25 THEN
        CALL PrintHeadings
    END IF
    INPUT ...
LOOP
END
'Subroutines and procedures are coded here
SUB PrintHeadings
 .
 .
 .
END SUB
```

过程通过 CALL 语句来调用，而子程序通过 GOSUB 语句来调用执行。两者都称之为子程序或模块。虽然它们有一些共同点，但两者本质上是不同的，本节我们只讨论两者的共同点，而把不同点放在下一节去讨论。过程同子程序一样，可以执行任意多次，CALL 语句和 GOSUB 语句一样，可以依据给定条件的满足与否来选择执行。最后，当过程执行至 END SUB 语句时，将返回到主程序中调用过程的 CALL 语句的下一条语句。也就是说，过程中的 END SUB 同子程序中的 return 语句起着同样的作用。

过程可以调用别的过程，如下所示：

```
SUB ReadAndProcess
    .
    .
    .
    LineCtr = LineCtr + 1
    IF  LineCtr > 25 THEN
        LineCtr = 0
        CALL PrintHeadings  'A CALL within a procedure
    END IF
    .
    .
    .
END SUB

SUB PrintHeadings
    .
    .
    .
END SUB
```

与前面子程序所讨论的类似，过程同别的子程序一样，应当在主模块的 END 语句之后才开始编写，另外，较重要的过程先编写，次要的过程后编写。举例来说，过程 ReadAndProcess 调用过程 PrintHeadings，那么过程 ReadAndPrecess 就应当先编写。有一点要注意，过程标志如 SUB PrintHeadings 是没有标点的，仅仅有一个冒号放在子程序名的后面。

我们把含有 CALL 语句的程序称作调用模块（calling module），而把过程本身称作被调用模块(called module)，调用模块通常是程序主体或者主模块，但有时也可以是次要模块或者子程序（过程和子程序）。在这一章，术语"主模块"和"程序主体是可以互换的。

§11.2.2 利用过程传递参数

在使用 GOSUB 的模块和使用过程的模块之间有一个很大的不同点，那就是对变量的处理。利用过程，我们可以通过参数（Parameters）来传递主程序或调用模块的变量，而且能够把参数返回调用模块。通过这种途经，我们可以显式地说明用于特定模块的变量以及用于主程序的变量。也就是说，过程，使得我们可以不管对数据的具体操作而把它模块化或结构化。

考虑这样一个程序，它要打印标题、细目以及总和。使用过程的程序可以在需要的时候分别调用打印标题模块、打印细目模块以及打印总和行模块，但这样一个程序同样可以分别在各个模块中对需要处理的数据进行显式说明(参见下页调用关系图)：

传递给过程或从过程中传出的变量都称作参数（Parameters），在过程中使用参数有下列优点：

① 可以通过CALL语句直接把数据传递给它们；

② 与主模块中的变量完全独立。也就是说，有些变量或参数当作全局变量（global variables）——在整个程序中都起作用；而另外一些变量或参数称作局部变量（local variables）——只在特定的模块中起作用。

```
                                'Main program goes here
                                 .
                                 .
                                 .
                        -------> CALL PrintHeadings (DateIn$, PageNo)
(Date$                           .
and PageNo                       .
are passed)                      .
                        -------> CALL PrintDetailLine (CustNo, CustName$, BalDue)
                                 .
      (CustNo,                   .
      CustName$,        -------> CALL PrintTotals (TotalAmt, TotalNoOfRecs)
      and BalDue                 .
      are passed)                .
                                 .
                                 END
                                 'Procedures go here
                        -------> SUB PrintHeadings (DateIn$, PageNo)
                                 .
                                 .
                                 .
                                 END SUB
                        -------> SUB PrintDetailLine (CustNo, CustName$, BalDue)
                                 .
      (TotalAmt and              .
      TotalNoOfRecs              END SUB
      are passed)                .
                                 .
                        -------> SUB PrintTotals (TotalAmt, TotalNoOfRecs)
                                 .
                                 .
                                 END SUB
```

其中：调用PrintHeadings模块时，Date＄和PageNo在主模块与PrintHeadings模块间传送。

调用PrintDetailLine模块时，CustNo、CusrName＄和BalDue在主模块与PrintDetailLine模块间传送。

调用PrintTotals模块时，TotalAmt和TotalNoOfRecs在主模块与PrintTotals模块间传送。

与此相反，GOSUB语句不能传递参数。也就是说，主程序中的所有变量都能在子程序中访问和修改，数据并不是针对某一个特定的模块的。与此类似，主程序也能够修改任何子程序所用的变量。因而使用子程序，将使我们冒很大风险，因为修改数据可能对主程序或别的模块产生不良影响。另外，由于数据的非模块和非结构化，其程序也是非结构化的，这导致了它在调试、维护以及修改方面的困难。

因此，过程能有效地改进大型复杂程序的设计，我们可以通过把要传递给过程的参数和从过程传回的参数严格区分开来提高其模块性。

另外，过程允许我们在同一个程序中的不同点使用一组指令来调用过程处理不同的数据。

§ 11.2.2.1 定义参数表

通常，要求一个过程去处理程序主体所产生的数据。也就是说，我们要求从程序主体按类传送数据给过程而不考虑什么时候过程被调用。为了实现这个功能，我们给过程建立了一个参数表如下所示：

```
    .
    .
    .
    PRINT "Enter Date of Transaction (Month/Day/Year)"
    INPUT DateOfTrans$
    CALL PrintHeadings (DateOfTrans$)──┐   ┌ The inputted date of transaction is
END                                    │  <  passed to the PrintHeadings
SUB PrintHeadings (DateOfTrans$) ←─────┘   └ procedure when it is called
    .
    .
    .
    PRINT "Date of Transaction is "; DateOfTrans$
    .
    .
    .
END SUB
```

其中：调用PrintHeadings模块时，交易的输入数据由主模块传入。

在上面的例子中，DateOfTrans $ 就是一个参数，它是一个变量，当过程被调用时，它就从程序的主体被显式地传送到过程中去。最多一次能从调用过程向被调用过程传送 16 个参数，就象上例一样，调用模块最常见的就是程序的主体。

传递给过程的参数表可以包含特殊值或者是直接量。

```
.
.
.
CALL PrintHeadings ("01/05/90")──┐   ┌ "01/05/90" is passed
.                                │   │ to the string variable
.                                │  <  DateOfTrans$ when the
END                              │   └ CALL is executed
SUB PrintHeadings (DateOfTrans$)←┘
```

其中：执行CALL语句时，"01／05／90"传入字符串变量DateOfTrans $。

§ 11.2.2.2 被调用模块中的参数应和调用模块中 CALL 语句所出现的参数个数相等

在 CALL 行和 SUB 行中必须包含相同的过程名以及相同的参数个数。

下列程序段是无效的

(1) 在CALL行和SUB行中出现的过程名不相同（Proc1相对于Procedure1):

```
Procedure1):
      :
CALL Proc1 (x, y, z) ──────┐
      :                     │
END                         │
SUB Procedure1 (x, y, z) ←──┘
      :
END SUB
```

(2) 在CALL语句中只有两个参数，而在SUB行中却有三个:

```
      :
CALL Procedure2 (a, b) ─────┐
      :                     │
END                         │
SUB Procedure2 (a, b, c) ←──┘
      :
END SUB
```

§11.2.2.3 在参数表中使用不同的名字

需要注意的是，在 CALL 语句和 SUB 语句的参数表中的参数名不一定相同:

```
:
INPUT DateOfTrans$
CALL PrintHeadings (DateOfTrans$) ──┐
:                                   │
END                                 │
SUB PrintHeadings (ProcessDate$) ←──┘
:
PRINT "Date of Run is ", ProcessDate$
:
END SUB
```

The value in DateOfTrans$ is passed to the variable called ProcessDate$ in the procedure

其中：执行CALL语句时，变量DateOfTrans$中的值由主模块传入过程中的字符串变量 ProcessDate$.

在上例中，DateofTrans$是CALL语句中的一个参数，它被送往PrintHeadings过程中的变量ProcessDate$，然后ProcessDate$就成了过程中代替DataOfTrans$中的变量名．在参数表中允许使用不同的名字，但有一点要注意，CALL语句和SUB行中的参数个数必须是相同的．如果有多于一个的参数要传递，它们是顺序传递的．

考虑如下情况：

```
CALL Proc1 (x, y, z) ──┐     ┌ x is passed to a
:                      │    {  y is passed to b
                       │     └ z is passed to c
END                    │
SUB Proc1 (a, b, c) ◄──┘
```

其中：x传到a；y传到b；z传到c；

以后我们会发现，有个因素使得程序员们有时在一个参数表中使用不同的名字．在CALL语句和SUB语句行中出现的参数名字是没有什么实际意义的，仅仅是参数表中变量的次序影响了参数的传递．

下面这段简单的编码，只是为了说明这一点：

```
First$ = "A"
Second$ = "B"
Third$ = "C"
CALL Proc1 (Third$, First$, Second$) ──┐
:                                      │
                                       │
END                                    │
'Procedures and subroutines go here    │
SUB Proc1 (First$, Second$, Third$) ◄──┘
    PRINT First$; Second$; Third$
    :

END SUB
```

最后打印出的结果是C A B，也就是说，是参数的顺序，而不是在CALL语句和SUB语句中的参数名的顺序决定了参数的传递：

调用模块中的参数顺序 **被调用模块中的参数顺序**

```
Third$      ————————→      First$
    (C. the value in Third$, is passed to First$)
First$      ————————→      Second$
    (A. the value in First$, is passed to Second$)
Second$     ————————→      Third$
    (B, the value in Second$, is passed to Third$)
```

其中：Third $ 中的值："C"传到First $；
First $ 中的值："A"传到Second $；
Second $ 中的值："B"传到Third $；

§ 11.2.2.4 传递参数到子程序，并且把结果返回到调用模块

到现在为止，我们重点举例说明了参数是如何通过 CALL 指令传递给过程或子程序的，并且显式说明了过程将如何对参数进行处理。这种显式说明数据传递的能力由于提高了程序的模块性从而使得其结构性大为加强。

有时，我们要以反方向传递参数；也就是说，从被调用模块传递参数给调用模块。

例 1:

```
DO  WHILE AreThereMoreRecords$ = "Yes"
    .
    .
    .
    CALL ReadARecord (CustNo, CustName$, BalDue, AreThereMoreRecords$)
    .
    .
    .
LOOP
END
SUB ReadARecord (CustNo, CustName$, BalDue, AreThereMoreRecords$)
    IF  NOT EOF(1) THEN
        INPUT #1, CustNo, CustName$, BalDue
    ELSE
        LET AreThereMoreRecords$ = "No"
    END IF
END SUB
```

在上述例子中，当过程执行后，CustNo、CustName $ 及 BalDue 三个输入区将从过程传递回主模块。初始化后，Are there More Records $ 中的值" yes"传递给 Read A Record，但若无更多的数据时，"No"将被返回到主模块.因此，参数表中应当包含被调用模块执行完后要回送给调用模块的那些变量。我们当然也可以双向传递变量，即：当过程被

调用时向它传递参数，而当过程运行完后返回参数。要实现双向参数传递，只需在参数表中增加一些参数用来存放要进行双向传递的值。

例 2:

```
:
INPUT Mo
INPUT Day
INPUT Yr
CALL CalculateDayOfWeek (Mo, Day, Yr, DayOfWeek)
:
PRINT "Day of Week Is ", DayOfWeek
:
END
SUB CalculateDayOfWeek (Mo, Day, Yr, DayOfWeek)
:
DayOfWeek = ...
:
END SUB
```

(Mo, Day, Yr is passed *from the main module to the procedure*)

(DayOfWeek is passed *from the procedure to the main module*)

其中：DayOfWeek由过程中传回主模块。
　　　Mo、Day、Yr由主模块传入过程。

执行了 CALL 语句后，Mo，Day，Yr 以及 Dav of week 被传给了过程。除了 Day ofweek 外，Day，Yr 及 Mo 都是输入区，执行 CALL 语句时都具有有效数据。过程运行后，四个变量都要返回主模块，而仅仅只变量 Day of week 的值被改变了，其实质就是只有变量 Day of week 的值被返回了。

因此，只用一个参数表就能实现：

(1)　向过程传递参数；
(2)　把数据返回到主模块或别的调用模块。

例 3:

```
'The first three variables are passed to the procedure when
 the call is executed

'NewBal is returned to the main module when the END SUB
'is executed
CALL UpdateTotal (BalOnHand, TotalWithdrawals, TotalDeposits, NewBal)
PRINT "New Balance is "; NewBal
:
END
```

```
SUB UpdateTotal (BalOnHand, TotalWithdrawals, TotalDeposits, NewBal)
    LET NewBal = BalOnHand + TotalDeposits - TotalWithdrawals
    :
END SUB
```

变量 BalOnHand、TotalWithDrawals 以人 TotalDeposits 中的值在运行时，传给了过程 VpdateTotal。这些值用来计算出 NewBal，并且在过程执行完后被返回主模块。

§11.2.2.5 过程的完整格式

我们已经知道，过程作为子程序，由标志符 SUB 起头，然后由终止标识符 END SUB 结束，其完整格式如下：

```
SUB 过程名[(参数表)]
    :
    [EXIT SUB]
    :
END SUB
```

过程名定义了过程的名字。过程名同子程序的标号一样，最多可以有 31 个字符，并且必须是唯一定义的。也就是说，过程名不能被程序中其它的过程或子程序用作程序名，也不能被当作变量名来使用。最好使用有意义的名字，但要尽可能的短，以减少由于名字太长而造成的输入错误。参数表是一个可选项，名项参数之间用逗号隔开，这些参数在过程被调用时可以向过程传递参数，而在过程执行完后可返回参数值。任何过程的参数表最多只能有 16 个参数。

Exit SUB 语句是用于非正常退出过程体的，其目的是避免处理大量无效的数据。

例:

```
SUB UpdatePayrollRec (EmpNo%, Salary)
    IF  Salary < 0 OR EmpNo% < 0 THEN
        PRINT "Invalid Data"
        EXIT SUB
    ELSE
    :
    END IF
END SUB
```

在子程序中是不允许使用 Exit 语句的，这是过程是一种比子程序更加模块化的工具的另一个原因。过程可以放在程序中的任何位置。通常跟子程序一样，放在主模块后面。编码时，重要的模块先编，次要的模块后编。

过程运行完后，控制返回到主模块中 CALL 语句后面的语句.

通常用术语子程序或模块来表示过程和子程序，而用主模块表示程序的主体.

自我测验(一)

1. (T 或 F) 过程和子程序都是编制模块化程序的工具.
2. (T 或 F) 自顶向下的程序比别的程序易于设计和阅读.
3. 子程序由_______语句来调用.
4. 子程序由_______开头，_______结束.
5. 子程序最好在程序的_______开始编程，在_______语句之后.
6. 过程是由_______语句来调用执行的.
7. 过程的起始标识符是_______，终止标识符是_______.
8. (T 或 F) 过程和子程序都能被别的过程和子程序调用执行.
9. (T 或 F) 过程与子程序相比有一个优点，即过程能显式说明调用模块传递给过程的数据.
10. (T 或 F) 过程与子程序相比有一个优点，即过程能用Exit语句，非正常退出过程.

§ 11.2.2.6 利用变参进行参数传递

有两种方法可将数据传递给过程，一种叫做变参(或引用)传送；另一种叫按值传送.

使用变参传递参数的方法是一种双向传递．也就是说，在调用模块中定义的参数传递给被调用模块，而在被调用模块中对这些参数进行修改后，其结果将返回到调用模块．使用按值传递参数的方法是一种单向传递也就是说，要传递的参数，如直接量、表达式以及变量等，只以一个方向进行传递，即从调用模块送往被调用模块．我们将详细讨论这两种参数传递方法，本节重点讨论变参传递方法.

使用变参进行参数传递，被调用模块和调用模块中的参数表可以含有相同的变量名字．如果在被调用模块中对这些参数进行了修改，这种变化将被返回到调用模块.

```
'Main Module goes here
INPUT A, B

PRINT "The Original Numbers Are "; A; B
CALL Switch (A, B)
PRINT "The Numbers Now Stored in the Main Module Are "; A; B
.
.
.
END
SUB Switch (A, B)
    Hold = B
    B = A
```

```
    A = Hold
    PRINT "The Numbers in Reverse Order Are "; A; B
END SUB
```

如果 A 输入的是 15，B 输入的是 10，那么三行打印的结果是：

```
The Original Numbers Are 15 10
The Numbers in Reverse Order Are 10 15
The Numbers Now Stored in the Main Module Are 10 15
```

可见，在 Switch 过程中 A 和 B 的内容被交换了，在被调用模块中对这些变量的修改在被调用模块执行完后返回给了调用模块。

即使在调用模块和被调用模块中使用不同的参数名，上例中的结果仍是完全一样的。

```
'The Main Module Goes Here
    INPUT A, B
    PRINT "The Original Numbers Are "; A; B
    CALL Switch (A, B)
    PRINT "The Numbers Now Stored in the Main Module Are "; A; B
END
SUB Switch (X, Y)
    Hold = Y
    Y = X
    X = Hold
    PRINT "The Numbers in Reverse Order Are "; X; Y
END SUB
```

如果 A=15，B=10，即使 SUB Switch … END SUB 模块中的参数用 x 和 y 来代替了 A 和 B，其最终的打印结果仍和上例完全相同。

```
The Original Numbers Are 15 10
The Numbers in Reverse Order Are 10 15
The Numbers Now Stored in the Main Module Are 10 15
```

由上可以看出，变参传递法，既可以向模块中传递参数，也可以从模块返回参数，或者双方向进行，这取决于参数的定义位置：

```
A = 5
B = 10
CALL SumIt (A, B, C)
PRINT C
END
SUB SumIt (A, B, C)
    C = A + B
END SUB
```

A, B are passed to SumIt
C is passed from SumIt

Passes C to the main module and then returns control to the statement after the CALL in the main module

其中: A, B被传入SumIt模块; C为SumIt模块的返回值. SumIt 模块将 C 返回到主模块, 并将程序流程转向主模块中 CALL 语句之后.

同样, 参数的传递不受参数表中变量名的影响:

```
A = 5
B = 10
CALL SumIt (A, B, C)
PRINT C
END
SUB SumIt (X, Y, Z)
    Z = X + Y
END SUB
```

A is passed to X;
B is passed to Y

Z is passed to C at the END SUB

其中: A被传到X; B被传到Y; 在 END SUB 处Z被传回C;

被调用模块及调用模块中的参数表何时使用相同的名字, 何时使用不同的名字. 当程序中只有一处调用该模块或者每次调用都处理同样的数据时, 参数表中可以含有相同的变量名.

例 1:

```
AreThereMoreRecords$ = "Yes"
CALL ReadARecord (CustNo, CustName$, BalDue, AreThereMoreRecords$)
DO WHILE AreThereMoreRecords$ = "Yes"
    .
    .
    .
    CALL ReadARecord (CustNo, CustName$, BalDue, AreThereMoreRecords$)
LOOP
END
SUB ReadARecord (CustNo, CustName$, BalDue, AreThereMoreRecords$)
    IF EOF(1) THEN
        AreThereMoreRecords$ = "No"
    ELSE
        INPUT #1, CustNo, CustName$, BalDue
    END IF
    .
    .
    .
END SUB
```

此处使用相同的参数表是为了将输入参数返回到被调用模块. CustNo, CustName $ 以及 BalDue 三个变量的值从被调用模块返回到调用模块, 如果满足结束条件, 变量 AreThereMoreRecords $ 值的变化立即从被调用模块返回到调用模块. 如果某个过程完成能适用于多组数据的系列操作, 那么最好的在参数表中使用不同的名字.

例 2:

```
:
CALL CalculateSum (BalDue, TransAmt, NewBal)
    :
CALL CalculateSum (QtyOnHand, PartsOrdered, NewQtyOnHand)
    :
END
SUB CalculateSum (X, Y, Z)
    Z = X + Y
END SUB
```

在这个例子中，过程 CaculateSum 用来处理两组不同的数据–结欠金额及库存. 第一次调用，NewBal 等于 BalDue 和 TransAmt 相加，最后，Z 被传回了主模块中的 NewBal. 第二次调用，作用是将变量 NewQtOonHand 赋值为 QtyOnHand 和 PartsOrdered 的和.

在这种情况下，最好使用不同的变量名，因为一个过程用来处理两组完全不同的变量. 对过程 caculate sum 的所有调用，在调用参数表中的第一个参数都被传给了 x，第二个参数都传给了 y. 在过程中求出的 Z 的值，传回到了参数表中的第三个参数. 有一点要注意，如果在过程体中对 x 和 y 作了修改，这种变化将被传给 Bal Bue 及 TramsAmt 或者是 Qty On Hand 及 ants Ordeved，具体传递给谁将取决于哪一次调用.

总之，在使用变参传递参数时，在被调用模块中参数的任何变化都将影响到调用模块中相对应参数，而不论在 CALL 语句和 SUB 语句中所使用的变量名.

§11.2.2.7 用按值传送的方法进行参数传递

通常使用按值传递参数的方法把直接量、表达式以及变量传送给过程，这种传送是单向的. 也就是说，按值传送是一种单向传送，从调用模块送往被调用模块. 下面先看一看直接量的传送情况:

一. 用按值传送的方法传送直接量

例 1:

```
CALL CalculateDayOfYear (12, 15, 90)
    :
END
```

```
SUB CalculateDayOfYear (Mo, Da, Yr)
     .
     .
     .
END SUB
```

例 2:

```
'CALL increases inputted salaries by 500 using the
'Pass-By-Value Method
CALL SalaryIncr (500)
     .
     .
     .
END
SUB SalaryIncr (Amt)
    INPUT Salary
    NewSalary = Salary + Amt
    PRINT NewSalary
END SUB
```

因此，使用按值传送的方法来传送直接量给被调用模块将不影响调用模块中的任何变量，这是与双向传送的变参传送所不同的地方。

二. 用按值传送的方法传递表达式

按值传送的第二个用处是把表达式送往被调用模块而不影响表达式中的操作数.
考虑下列情况:

```
'Passing an expression to a called
'module is a one-way transmission
CLS
I = 1
CALL DoubleIt (2*I)          '2*I is an expression
PRINT I
END
SUB DoubleIt (I)             'I = 2 in the procedure only
    PRINT I
END SUB
```

打印结果为:

2 ← 过程PoubleIt中I的值.
1 ← 调用过程前主模块中I的值.

之所以有这种单向传送存在是由于从主模块传送给被调用模块的参数是一个表达式(2 * I)，而不是一个变量.

三. 用按值传送的方法向被调用模块传送变量

最后，可以使用按值传送的方法从调用模块向被调用模块传递变量，这种传送也是单向的。

为了把 CALL 语句中参数表的变量传递给被调用模块，而不会将参数的变化返回，需要在 CALL 语句中要进行单向传送的参数上外加一对圆括号。

再次讨论一个交换过程，在过程中的 A 与 B 的值进行了交换。但交换只是在过程中进行，而主模块中它们原来的值保持不变，请看下例:

例 1:

```
CLS
A = 1
B = 2
PRINT A, B
CALL Switch ((A), (B))     'Note parentheses around A and B
PRINT A, B
END
SUB Switch (A, B)
    Hold = A
    A = B
    B = Hold
    PRINT A, B
END SUB
```

输出的结果是:

```
1  2
2  1
1  2
```

例 2:

```
CLS
A = 1
B = 2
PRINT A, B
CALL Switch ((A), (B))    'Note parentheses around A and B
PRINT A, B
END
'The parameter list can have any variable name
SUB Switch (X, Y)
    Hold = X
    X = Y
    Y = Hold
    PRINT X, Y
END SUB
```

输出的结果是:

1 2

2 1

1 2

在同一个 CALL 语句中既可以按值传送，也可以用变参传送

在同一个 CALL 语句中可以用上述两种方法进行参数传递。也就是说，有的参数可以按值进行传送，而另外一些则用变参进行传送。这样，有些参数在过程体中的变化将影响主模块中的变量，而另外一些参数在过程体中的变化不影响主模块中的任何变量；请看下例:

```
INPUT Salary1
CALL Proc1 (1500, (Salary1), Salary2)
PRINT "Salary1 = "; Salary1
PRINT "Salary2 = "; Salary2
END
SUB Proc1 (Increase, Sal, Sal2)
    Sal = .50 * Sal
    Sal2 = Sal + Increase
END SUB
```

下面是运行结果:

? 1800

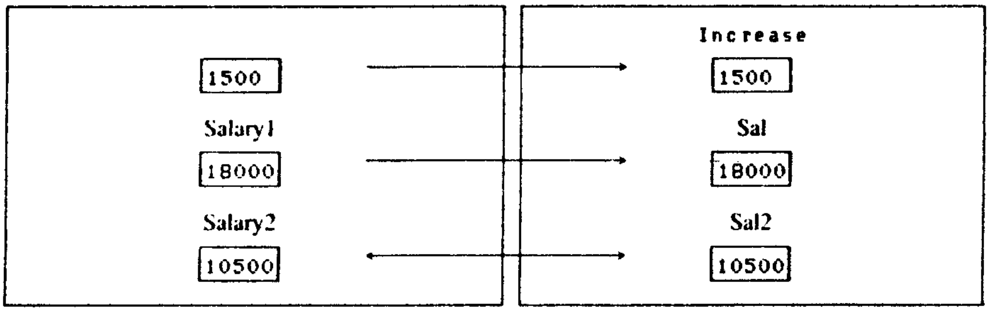
Slary1 = 1800

Slary2 = 10500

主模块和过程间的数据是按如下方式传送的:

Main Module
CALL Proc1(1500, (Salary1), Salary2)

Procedure
SUB Proc1 (Increase, Sal, Sal2)
⋮
END SUB



Salary1 利用按值传送的方法传递给 Sal1．因此，在过程中对 Sal1 的变化将不会影响 Salary1 的值；Salary2 是用变参进行传送的．

概括起来就是：按值传送用于向被调用模块传送直接量、表达式、或者要求过程中参数的变化不影响调用模块中的初始参数时．如果要调用模块中的参数不受影响，只需把变量用一对圆括号括起来就行了，这样 Turbo Basic 将它作为一个值来处理，因而采取了按值传送的方法．

§ 11.2.2.8　数组传送

数组可以象下面那样传递给被调用模块：

```
DIM  Array1(1:10)
      :
CALL  ArrayProcess(Array1( ))
```

SUB 语句将类似地用空括号来说明：

```
SUB  ArrayProcess  (Array1  (  ))
```

上例中的数组传送是用变参传送的方法．

小　　结		
举　例	所用的方法	源参数可改变否?
CALL　Proc(A, B, C)	按值传送	能
CALL　Proc(5, 3, 0)	按值传送	不能(它们为常数)
CALL　Proc(5 * I)	按值传送	不能．表达式，如 5*I 不改变调用模块中的变量 I.
CALL　Proc((A), (B), (C))	按值传送	不能．在过程中，它们当作数值处理．
CALL　Proc(ArrayX( ))	变参传送数组	能

字符串变量、直接量以及数组与数值变量的传送方式相同．

下一章我们将讨论用于子程序中、而不是参数的变量情况；例如：

```
SUB  Proc1(x)
        y  =  0        ' y不是参数
        :
        :
END  SUB
```

下章将解释为什么象 y 这样的变量既可以定义为局部变量，又可以定义为全局变量。

自我测验(二)

1. 过程执行完后，控制将返回到________。
2 (T or F) 一个过程能调用另外一个过程。
3 过程由________语句开始，________结束。
4 (T or F) 最好在程序的末尾再编写过程。
5 (T or F) 如果A 过程调用B过程，那么B过程必须先出现。
6 包含Call 语句的程序称作________。
7 考虑下列过程:

```
SUB SumTheSquares(X, Y)
X = X ^ 2
Y = Y^2
Sum  = x+Y
PRINT  Sum
END  SUB
```

指出下列情况下的结果:

```
a. CALL  SumTheSquares (5, 9)
b. CALL  SumTheSquares (3+2, 3^2)
c. First  = 3
   Second  = 4
   Call  SumTheSquares (First , Second)
d. y = 1
   x = 4
CALL SumTheSquares(x, y)
```

8 写一个过程打印输入变量x值的3倍而不影响x的值。
9 写一个过程打出输入变量x的三倍值，并且要求x中也成了原值的三倍。
10 考虑下面的数字序列:

1
1
2
3
5
8

13
21
34

第三个数是前面第一第二个数的和，第四个数是第二第三两个数的和，依此类推。编写一个程序，调用过程来打印这个序列，直至打到数字超过 100.

11 用过程编一个短程序，打印下面这首歌曲:

School is almost over and I'm glad
School is almost over and I'm glad
But when I think about final exams
I just get mad
Yes, Sir
I just get mad

12 使用子程序重写第11题.

13 指出下面程序的运行结果:

```
X = 1
Y = 2
PRINT X, Y
CALL Test (X, Y)
PRINT X, Y
END
SUB Test (X, Y)
    X = Y
    Y = 0
    PRINT X, Y
END SUB
```

14 指出下面程序的运行结果:

```
BigNum = 5
LittleNum = 10
PRINT BigNum, LittleNum
CALL Switchem (BigNum, LittleNum)
PRINT BigNum, LittleNum
END
SUB Switchem (First, Second)
    Hold = First
    First = Second
    Second = Hold
END SUB
```

15 有如下过程:

```
SUB SquareSum (First, Second, Sum)
    First = First ^ 2
    Second = Second ^ 2
    Sum = First + Second
PRINT "Square Sum", First, Second, Sum, Answer
END SUB
```

指出下面程序的运行结果:

```
First = 1
Second = 2
CALL SquareSum (First, Second, Sum)
PRINT "Main", First, Second, Sum, Answer
CALL SquareSum (Second, First, Answer)
PRINT "Main", First, Second, Sum, Answer
```

16 假定有下列所示程序:

```
CALL GetNumbers(Num1, Num2)
CALL SwitchEm(Num1, Num2)
CALL PrintTheResults(Num1, Num2)
```

程序与用户的交互如下:

```
What is the first number ?
85
What is thd secnd number ?
63
The numbers in reversed order are 63 and 85
```

请编写出上述三个过程.

17 请指出下列程序的运行结果:

```
    X = 1
    Y = 2
    Z = 3
    W = 4
    PRINT X, Y, Z, W
    CALL Change (Y, X, W)
    PRINT X, Y, Z, W
END
SUB Change (Z, X, W)
    X = 5
    Y = 6
    Z = 7
    W = 8
    PRINT X, Y, Z, W
END SUB
```

18 编一个程序接收雇员名和支付率．在过程中计算支付额20%的增值，并将这个值返回给主程序．编一个输出文件内有雇员名和新的支付率．

19 经理决定给全体职工发一次性的奖金，利率为15%，请编一个程序接收职工名字、支付率以及工作时间．在一个过程中计算实发数，其中包括奖金，并打印出结果．在主程序中写一个输出文件，其中包括职工名字以及不含奖金的应付款．

20 经理决定将19题中的奖金变为永久性的，修改程序以使输出文件中的奖金为永久性的．

§11.3 过程及子程序小结

§11.3.1 何时使用子程序

1. 如果程序中的某指令序列要不止一次地运行，最好将这组指令编成子程序或模块．在必要的地方调用.在我们所说的子程序或模块，既可以是过程，也可以是子程序．
2. 在诸如初始化变量，打印数据（标题、细目及汇总）、处理函数以及任务结束等例程很复杂时，建议使用模块或子程序来编写．
3. 如果一组相关指令是否被执行要取决于特定条件的满足与否，那么应将这组指令编成子程序．

§11.3.2 子程序应是内相关的

如果程序中的模块分别包含了一些相关的任务，而不同模块之间的任务是不相关的，那么这个程序应认为是内相关的（cohesion）．内相关程序是这样的一种程序，程序内部各模块之间是相互独立的．这个概念类似于某种文章，句子组成了段，而段与段之间是相互独立的．

概括地说就是，子程序是一种使程序更加模块化和自顶向下结构化的途经．通过把复杂而频繁使用的指令编成内相关过程，可以更好地设计程序，使之易于阅读、调试及在必要的时候进行修改．

§11.3.3 何时使用子程序，何时使用过程

当子程序使用完全独立的变量，即：①主程序不向它传递参数;②不影响主程序中的变量，此时过程与子程序对数据的处理方式相同．如果子程序需要从主程序向它传送参数，并且要向主程序返回参数，最好使用过程．如果子程序中需要异常出口，应使用过程．虽然我们在至今为止的所有模块顺使用了子程序，但从更好的观点来看，应该使用过程．它由于显式指明数据的传送方式而使编程更加模块化，而且使得程序更加结构化．设计得更好．

§ 11.4 层次图是自顶向下模块化程序的规划工具

如前所述，使用伪代码来设计程序可以使结构化设计的概念得以正确而有效的实施．但对精心设计的程序的另一种主要方法，此处指的是自顶向下的方法又如何呢? 需要一种规划工具用来说明程序中模块之间的自顶向下的关系．在复杂而冗长的程序里它将是非常有用的．因为程序中模块之间的相互调用使得我们很难跟踪模块之间的相互关系．这种工具将用于包含子程序或过程的程序中．

用于说明程序的自顶向下设计方法的最佳工具是层次图或结构图．层次图或结构图以图形的方式说明了程序中子程序或模块之间的相互关系．当然，子程序或模块既包括子程序，也包括过程．程序中有一个子程序或多个子程序调用别的子程序时, 层次图用于记录其相关关系．它的主要目的是提供一种直观的方法来说明程序中的模块是如何与别的模块发生关系的．利用层次图，可以在编程前将某一个完整的任务分解成许多易于管理的子功能块，并且在编码时能够记录这些子功能段的相互关系．

因此，在编程序前，需要进行两方面的逻辑设计:

(1) 用伪代码来设计逻辑结构；也就是说，指令是如何运行的．

(2) 用层次图来说明在自顶向下方法下进行总体设计时模块之间的调用关系．

例:

在Turbo Basic 里，自顶向下程序设计或结构化程序设计的概念也就是：先编主模块，后编次要模块。

一个主模块或者程序被细分成称之为子模块的组成部分．自顶向下的设计就象是一页纸的轮廓，起先用铅笔写上主标题及其组成部分，在总体组织完成之后，然后着重讨论细节部分．

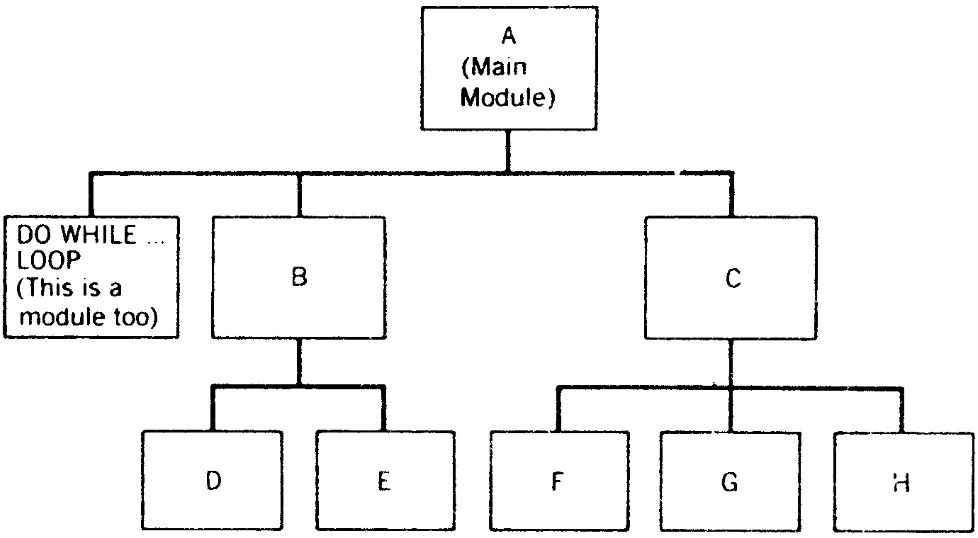
请看下列关于层次图的几点说明:

层次图

1. 层次图是一个将程序模块表示为方块，并用连接线描述模块间相互关系的结构图．
2. 模块是一个实现特定功能的设计良好程序段．

下例说明了层次图中模块之间的关系．实际上，应该使用有意义的名字，此处使用字母 A～H 是为了使所代表的标号或过程名简短一些，并突出最重要的概念．

层次图的样例



注意：字母A到H既可能表达了能被GOSUB语句调用执行标号，也可能是能用CALL语句调用执行的过程。

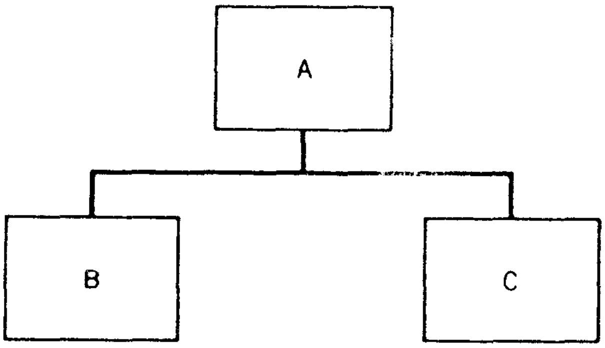
考虑下例：

```
'Main Module with DO WHILE ... LOOP goes here
     .
     .
     .
     CALL B
         .
         .
     CALL C
         .
         .
         .
END
'Subprograms--procedures and subroutines--go here
SUB B
     .
     .
     .
     CALL D
     .
     .
     .
     CALL E
     .
     .
     .
END SUB                    'End of Sub B
SUB C
     .
     .
     .
     CALL F
     .
     .
     .
     CALL G
```

```
        .
        .
        .
        CALL  H
    END SUB                    'End of Sub C
```

层次图说明了模块之间的相互关系。它不是象伪指令那样用来说明具体指令的。层次图中的每个块或每个方框都代表了一个模块，如一个 Do while 循环，一个子程序或者过程。如果一个模块调用另一个模块。它将用一个单独的方框表明而不管它是一个子程序还是一个过程。

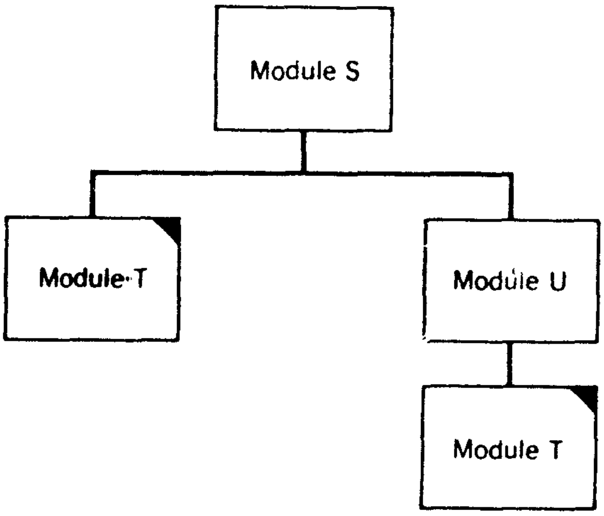
考虑如下层次图的前面部分:



从这个摘录中可以看出，模块 B 和模块 C 都是由模块 A 来调用执行的。

尽管某个模块可能是用 CALL 语句来调用执行的，模块本身也可能含有 CALL 语句。例如，模块 D 是由模块 B 来调用的，而模块 B 又是在主模块 A 中调用执行的。

考虑下述层次图，它与前面又有所不同:



这个层次图表明模块 T 在两个地方被调用，从模块 S 和 V 中两处调用。为了突出模块 T 在程序中不止一处调用，我们将代表模块 T 的方框角上涂黑。

概括起来就是：层次图说明了模块之间的关系，哪一个模块从属于另外的模块，及某个模块在程序中是否不止一次被调用执行了。这种层次图使得记录程序中的逻辑关系变得很容易。另外，如果某个模块以后需要进行修改，层次图能向人们提示修改将会怎样影响到程序的其它部分。它并不考虑模块中的具体指令，而只是模块的关系。具体的指令序列用伪指令来说明，伪指令与层次图合在一起成为一种程序设计工具。

层次图有时也称作可见目录（Visual Table of Contents），因为它提供了一个程序的图形说明。参见图 11.1 所示的 Turbo Basic 程序，它用来为每个雇员计算工资，并且超时部分以 1.5 倍计算。程序设计成每页打印 25 行，然后换页继续打印标题及内容。

```
CLS
CALL GetData (EmployeeName$,PayRate,HoursWorked)
PageNo = 0
NewPage$ = CHR$(12)
CALL PrintHeadings (NewPage$,LineCount,PageNo)
DO WHILE PayRate <> 0
        CALL ComputeWages (PayRate,HoursWorked,TotalWages)
        LPRINT TAB(10);EmployeeName$;
        LPRINT TAB(30);
        LPRINT USING "$$#,###.##";TotalWages
        LineCount = LineCount + 1
        IF LineCount > 24 THEN
                CALL Printheadings (NewPage$,LineCount,PageNo)
        END IF
        CALL GetData (EmployeeName$,PayRate,HoursWorked)
LOOP
PRINT "End of Run"
END
SUB GetData (EmployeeName$, PayRate,HoursWorked)
        PRINT "Enter Employee Name: ";
        INPUT EmployeeName$
        PRINT "Enter Pay Rate: ";
        INPUT PayRate
        PRINT "Enter Number of Hours Worked: ";
        INPUT HoursWorked
END SUB
SUB PrintHeadings(NewPage$,LineCount,PageNo)
        PageNo = PageNo + 1
        LPRINT NewPage$
        LPRINT TAB(10);"Page";
        LPRINT TAB(15);
        LPRINT USING "##";PageNo
        LPRINT
        LPRINT TAB(7);"Employee Name";
        LPRINT TAB(30);"Total Pay"
        LPRINT
        LineCount = 0
END SUB
SUB ComputeWages (PayRate,HoursWorked,TotalWages)
        IF HoursWorked <= 40 THEN
                OvertimeWages = 0
                RegularWages = PayRate * HoursWorked
        ELSE
                CALL OvertimeProcedure(PayRate,HoursWorked-40,OvertimeWages)
                RegularWages = 40 * PayRate
        END IF
        TotalWages = RegularWages + OvertimeWages
END SUB
SUB OvertimeProcedure (PayRate,Hours,OvertimeWages)
        OvertimeWages = Hours * PayRate * 1.5
END SUB
```

图 11.1 工资单程序样例

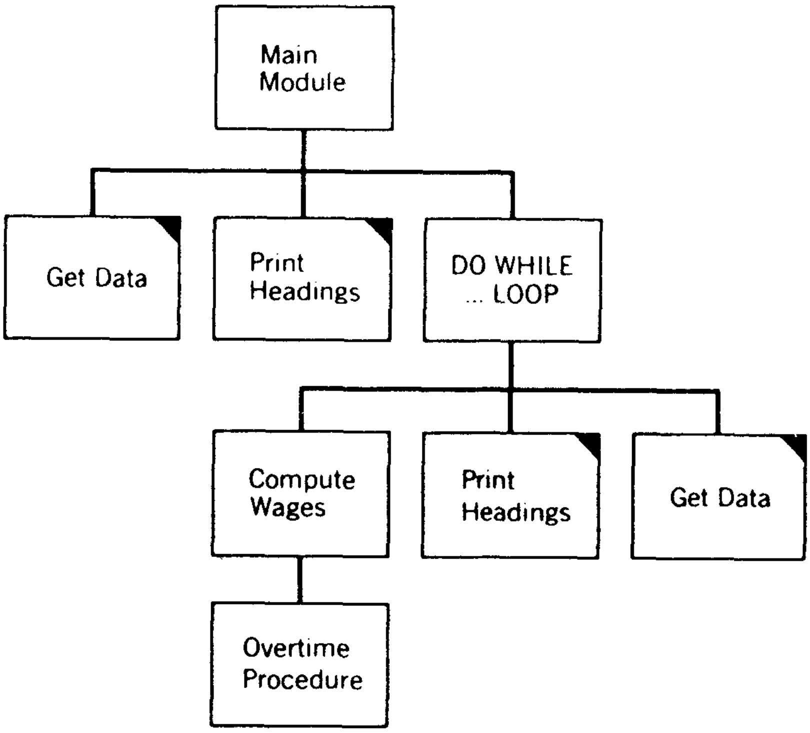


图 11.2 工资单程序样例的层次图

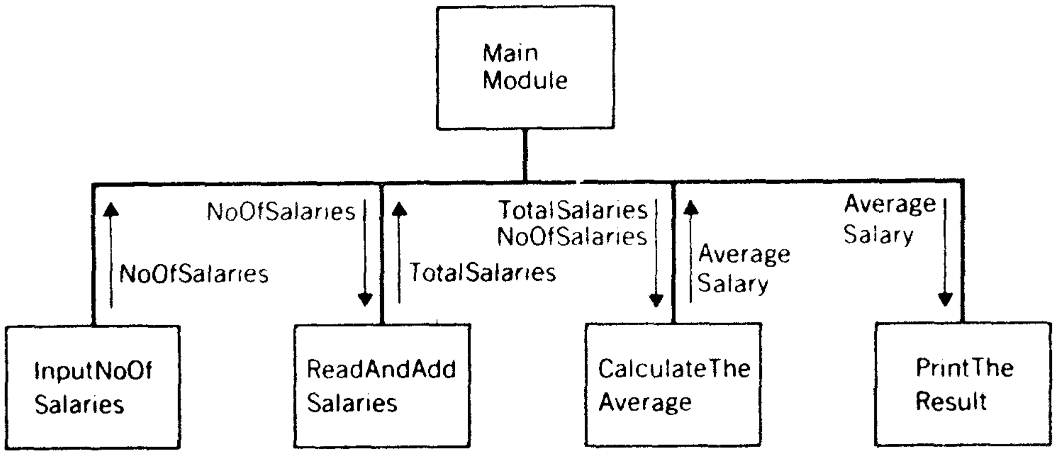
这个发工资程序的层次图或 VTOC 如图 11.2 所示．注意层次图直观说明了模块间的相互关系．标有黑三角的模块在程序中有不止一处进行调用．Do While 循环被看作一个单独的模块．有一点要注意，当象 Overtime Procedure 这样从属的模块执行完后，控制将返回次高级模块；在本例中为 Compute Wages 执行完时，也就时说，没有更多的输入了．此时控制返回 Do while 循环模块．由于层次图或结构图使用这种层次观点来说明了程序的逻辑结构，因而它也是一种自顶向下程序设计的工具．

由于能向过程传递参数以及从过程返回参数，因而我们在说明过程的层次图或结构图上加上数据流信息．假如要编一个程序来打印给定某星期所支付的工资总数及平均工资，需要建立如下几个独立过程：

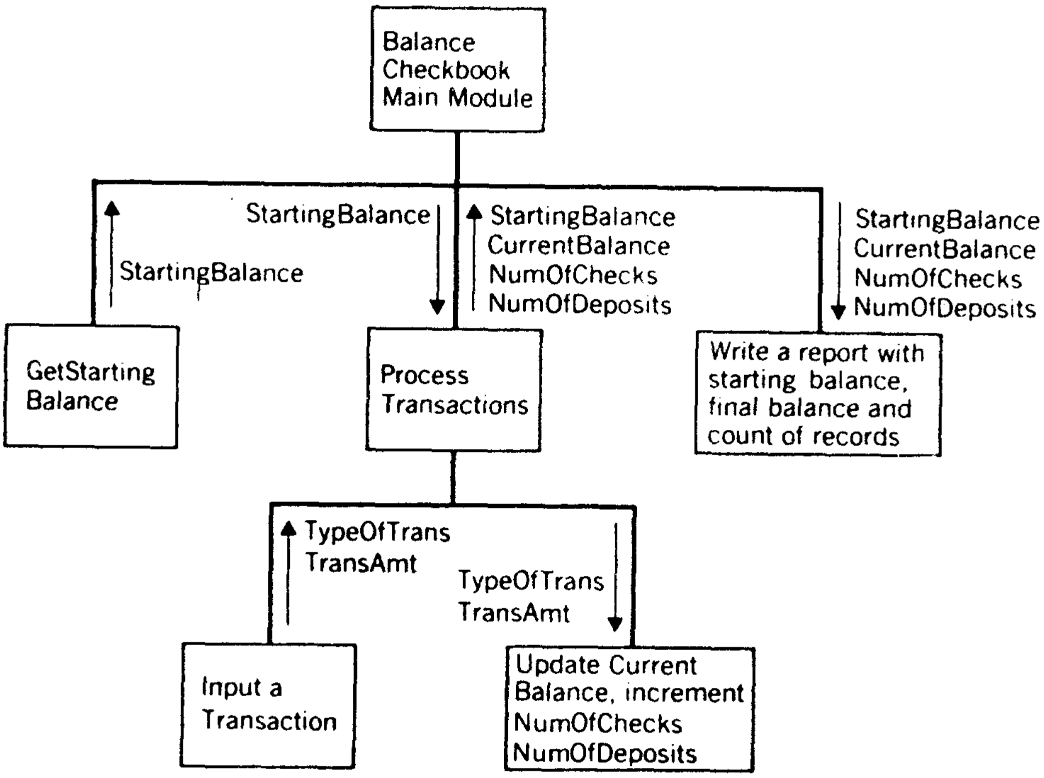
InputNoOfSalaries
ReadAndAddSalaries
CalculateTheAverage

PrintTheResult

这个程序的层次图将包含如下所示的数据流信息:



加上参数传递的概念，考虑下面关于个人支票本收支平衡情况程序的层次图:



总之，层次图有下面几个用处:

层次图或结构图的用处

1. 可帮助程序员、系统分析员和用户查看模块间的联系。
2. 可帮助程序员调试和修改程序。
3. 可帮助用户弄清程序中的逻辑流程；层次图或结构图可能包含数据流程信息。
4. 可帮助程序管理人员评估程序的效能。

因此，层次图同伪程序一样，是一种规划和归档工具。

编制一个又大又复杂的程序时，建议用户先画出层次图，再写出其伪代码。这些工具在制订程序的逻辑结构时是非常有效的。尽管在编写较简单程序时，没有很复杂的逻辑控制结构，因而用不上层次图，但培养了使用这种工具的习惯将对用户以后编制大型程序有好处。

§11.5 本章辅导

§11.5.1 本章小节

一、自顶向下模块化程序设计的回顾

1. 模块分层编写，这就是自顶向下模块化程序设计。
   (1) 使用自顶向下模块化方法精心设计的程序。
      a. 这种方法使得程序易于编写、调试以及修改；
      b. 模块可以独立开发设计
   (2) 自顶向下模块化程序减少了重复劳动
      a. 一个子程序只在一个地方编写后，可以在程序中的多个的地方进行调用，甚至在不同的程序中进行调用。
      b. 模块或子程序可以编成子程序或过程，也可以编成函数，这将在下一章进行讨论。
2 子程序可以用来实现自顶向下的编程技术
   (1) GOSUB语句可以在程序中的任何位置
   (2) 子程序减少了重复编码
   (3) 子程序应采用自顶向下的方法进行编制。
   (4) 按惯例，子程序应在END语句之后再进行编程 .

二、在自顶向下模块化程序中使用过程

1. 过程既模块化了数据传送方法又模块化了指令。

(1) 过程用CALL 语句调用执行.

(2) SUB...END SUB 部分定义了过程

(3) 过程同子程序一样，要在END语句之后再开始编程.

2 要传送给过程的数据及要从过程返回调用模块的数据被显式定义.

(1) 所有要传送的参数都要在CALL语句中按名列出

(2) CALL 语句应和与之相应的SUB语句含有相同的过程名及同样数目的参数个数

a. 参数是按其在参数表中的位置传送的

b. CALL 语句和SUB 语句中的参数名可以不相同

(3) 参数既能从调用模块传往被调用模块，也能从被调用模块传回调用模块，或双向进行

(4) Ex1t SUB 语句可以用来非正常退出过程.

3 变参传送数据的方法允许数据从调用模块传向被调用模块，并能返回数据.

(1) 如果在CALL语句和SUB语句中使用了相同的参数名，过程体中对参数的任何修改都将返回到调用模块.

(2) 如果过程适用于几组数据，那么在CALL语句和SUB语句中最好使用不同的名字（注意：值的任何变化都将传回调用模块而不管参数使用的名字).

4 按值传送方法把参数传给过程，而不能将参数传回到调用模块.

(1) 直接量能通过参数表由调用模块传给被调用模块

(2) 数字表达式能通过参数表由调用模块传给被调用模块

(3) 如果变量加上括号它们的值将被传给过程，但其任何变化都不能返回调用模块

(4) 在同一个CALL语句中既可以有变量参传送，也可以有按值传送.

三、子程序和过程作为子程序的小结

1. 程序中需要多处执行某一组指令时，最好将它编成子程序
2. 某一操作很复杂时，将它编成子程序
3. 要编制一个内相关程序，每个模块应完成一个独立的任务
4. 当需要传递数据时，过程优于子程序.

四、层次图作为自顶向下模块化程序的设计工具

1. 层次图以图形方式说明了子程序及模块之间的关系
2. 在自顶向下正模块化程序设计中层次图与伪代码一起来进行一个完整程序的设计
3. 层次图既是一种设计工具，也可以用于文本归档.

§ 11.5.2 本章自测

请看下面程序：

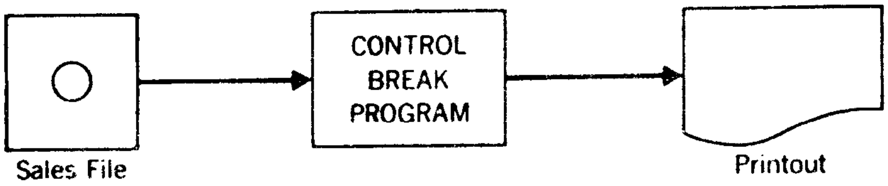
```
LET A = 15
LET B = 25
LET C = 40
PRINT A, B, C
CALL Switch1(A, B, C)
PRINT A, B, C
CALL Switch2((A), (B), (C))
PRINT A, B, C
END
SUB Switch1 (A, B, C)
    LET T = A
    LET A = B
    LET B = C
    LET C = T
END SUB
SUB Switch2 (A, B, C)
    LET T = A
    LET A = B
    LET B = C
    LET C = T
END SUB
```

1. 程序的输出结果是什么?
2. 解释一下打印的第一组及第二组数据为什么不相同?
3. 请解释为什么打印的第二、三组数据是相同的.
4. 当数据从调用模块传给被调用模块，以及从被调用模块返回调用模块时，这种数据传送方式称作________.
5. 如果数据只能从调用模块传给过程，这种参数传送方式称作________.
6. 在调用模块及过程之间传递的数据或者变量称作________.
7. 程序中的每个模块完成一个单独的任务，这个任务由许多相关的操作组成，则这样的程序称作________.
8. 是一种用于自顶向下模块化程序的设计工具.
9. 编制一个过程计算贴现率，当出售额大于500美元时，贴现率为15%，否则为10%. 请把贴现百分率、出售额及贴现值当成参数.
10. 请重编上述过程，唯一不同之处是此时贴现率不做为参数，也就是说不从调用模块传递.

§ 11.5.3 实例

考虑如下问题描述:

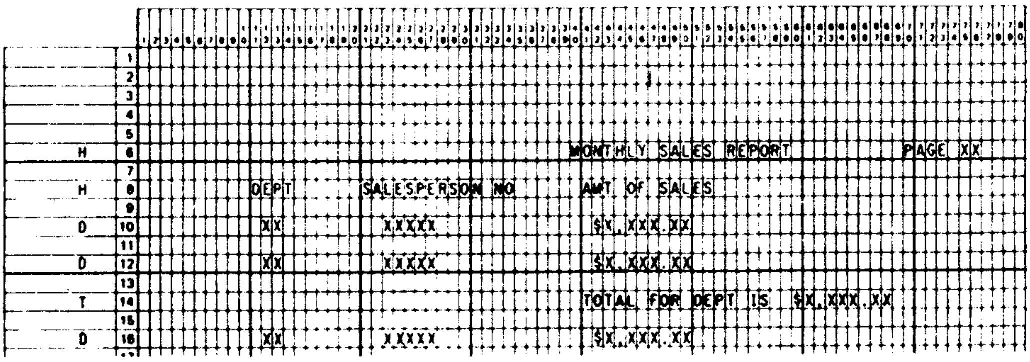
系统流程图



销售记录格式

DeptNo, SalesNo, AmtOfSale

输出清样的打印格式图



输入样例

```
1,12345,988.5499877929688
1,12346,3537
1,12347,34.9900016784668
2,12222,9877
2,12234,87.76999664306641
3,15645,980
3,12321,1987
4,12999,1343.300048828125
4,16732,1779
4,16437,4939.08984375
```

写一个程序实现如上所说的控制间隔打印，图 11.3 给出了答案及输出结果。

注意： 这个程序与第 7 章的图 7.6 相同，只是第 7 章采用子程序而不是过程。

其伪代码及层次图均分别与图 7.2 和图 7.5 相同。

```
'*************************************************************
' This is an example control break program that uses
' procedures as subroutines
'*************************************************************
CLS
CALL InitializeModule(AreThereMoreRecords$, FinalTotal, PageNo)
CALL ReadModule(DeptNo, SalesNo, AmtOfSale, AreThereMoreRecords$)
LET HoldDept = DeptNo
CALL PrintHeadings(LineCounter, PageNo)
'*********************************************************
'The DO WHILE determines if a control break has occurred
'and processes records accordingly
'*********************************************************

DO WHILE AreThereMoreRecords$ = "Yes"
        IF HoldDept <> DeptNo THEN
                CALL ControlBreak(DeptTotal, FinalTotal,_
                        DeptNo, HoldDept, LineCounter)
        END IF
        CALL DetailProcessing(LineCounter, AreThereMoreRecords$, DeptNo,_
                        SalesNo, AmtOfSale, DeptTotal, PageNo)
        CALL ReadModule(DeptNo, SalesNo, AmtOfSale, AreThereMoreRecords$)
LOOP
CALL ControlBreak(DeptTotal, FinalTotal, DeptNo, HoldDept, LineCounter)
LPRINT
LPRINT TAB(41); "THE FINAL TOTAL IS ";
LPRINT TAB(60);
LPRINT USING "$###,###.##"; FinalTotal
CLOSE #1
END

'*************************************************************
' The control break procedure produces totals when a change
' in DeptNo occurs
'*************************************************************
SUB ControlBreak (DeptTotal, FinalTotal, DeptNo, HoldDept, LineCounter)
        LPRINT
        LPRINT TAB(41); "TOTAL FOR DEPT IS ";
        LPRINT TAB(60);
        LPRINT USING "$#,###.##"; DeptTotal
        LPRINT
        LPRINT
        LET FinalTotal = FinalTotal + DeptTotal
        DeptTotal = 0
        HoldDept = DeptNo
        LET LineCounter = 99
END SUB
'*****************************************************************
' The DetailProcessing procedure prints output for each input
' record read
'*****************************************************************
SUB DetailProcessing (LineCounter, AreThereMoreRecords$, DeptNo,_
                SalesNo, AmtOfSale, DeptTotal, PageNo)
        IF LineCounter > 25 AND AreThereMoreRecords$ = "Yes" THEN
                CALL PrintHeadings(LineCounter, PageNo)
        END IF
        LPRINT TAB(12);
        LPRINT USING "##"; DeptNo;
        LPRINT TAB(23);
        LPRINT USING "#####"; SalesNo;
        LPRINT TAB(42);
        LPRINT USING "$#,###.##"; AmtOfSale
        LPRINT
        LET LineCounter = LineCounter + 1
        LET DeptTotal = DeptTotal + AmtOfSale
END SUB
```

```
'**********************************************************
' The initialize procedure initializes storage areas and
' opens files
'**********************************************************

SUB InitializeModule (AreThereMoreRecords$, FinalTotal, PageNo
        OPEN "Sales.Dat" FOR INPUT AS #1
        LET AreThereMoreRecords$ = "Yes"
        LET FinalTotal = 0
        LET PageNo = 0
END SUB

'**********************************************************
' The PrintHeadings procedure prints new headings at the
' beginning of the run, when a page overflow occurs or
' when a control break occurs
'**********************************************************
SUB PrintHeadings (LineCounter, PageNo)
        LPRINT CHR$(12)

        LET PageNo = PageNo + 1
        LPRINT TAB(40); "MONTHLY SALES REPORT";
        LPRINT TAB(70); "PAGE";
        LPRINT TAB(75);
        LPRINT USING "##"; PageNo
        LPRINT
        LPRINT TAB(11); "DEPT";
        LPRINT TAB(21); "SALESPERSON NO";
        LPRINT TAB(41); "AMT OF SALES"
        LPRINT
        LET LineCounter = 0
END SUB

'**********************************************************
' The ReadModule procedure reads input records and
' determines if an end of job has been reached
'**********************************************************
SUB ReadModule (DeptNo, SalesNo, AmtOfSale, AreThereMoreRecords$)
        IF NOT EOF(1) THEN
                INPUT #1, DeptNo, SalesNo, AmtOfSale
        ELSE
                LET AreThereMoreRecords$ = "No"
        END IF
END SUB
```

图 11.3(a) 实例的答案

```
                              MONTHLY SALES REPORT           PAGE  1

DEPT       SALESPERSON NO     AMT OF SALES

  1           12345              $988.55

  1           12346            $3,537.00

  1           12347               $34.99

                              TOTAL FOR DEPT IS  $4,560.54
```

```
                              MONTHLY SALES REPORT            PAGE  2
DEPT        SALESPERSON NO     AMT OF SALES
  2              12222           $9,877.00
  2              12234              $87.77

                               TOTAL FOR DEPT IS   $9,964.77
```

```
                              MONTHLY SALES REPORT            PAGE  3
DEPT        SALESPERSON NO     AMT OF SALES
  3              15645             $980.00
  3              12321           $1,987.00

                               TOTAL FOR DEPT IS   $2,967.00
```

```
                              MONTHLY SALES REPORT            PAGE  4
DEPT        SALESPERSON NO     AMT OF SALES
  4              12999           $1,343.30
  4              16732           $1,779.00
  4              16437           $4,939.09

                               TOTAL FOR DEPT IS   $8,061.39

                               THE FINAL TOTAL IS   $25,553.70
```

图 11.3(b) 输出样例

§11.5.4 复习题

一、综合题

下列各题均要求用过程完成:

1. 编一个程序段，打印如下图形:

```
**         **
***********
**         **
**         **
***********
**         **
**         **
***********
**         **
```

2. 编一个程序用于接收以米为单位的距离值，并将它转换成码、英尺、英寸。例如，如果输入的距离值为 2.5 米，打印结果为：

   2.5 米 ＝ 2 码，2 英尺，2.3 英寸

   对输入数据、计算及打印分别编制过程。

   转换关系如下：

   1 码 ＝ 0.914403 米
   1 英尺 ＝ 0.304801 米
   1 英寸 ＝ 0.0254001 米

3. 写一个程序用于打印如下的乘法表

Num	2x	3x	4x	5x	6x	7x	8x	9x	10x
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	20	30	40	50	60	70	80	90	100

4. 编一个程序对每班顺序输入20个学生，并打印班级平均。共有6个班要进行处理。
5. 编一个程序用于输入本金、利息率以及投资的年数，并打印综合利息表：

```
      Principal ___
      Interest ___

Year      New Principal      Accrued Interest
 1
 ⋮
 N
```

二、数据确认

修改实例使之能够:

(1) 检查出所有的错误;

(2) 打印出汇总控制表（处理的记录数、所遇到的错误个数、分类总数)。

§11.5.5 调试练习

指出下面程序中的错误:

1.

```
CALL ReadModule
  :
SUB ReadModule
    INPUT #1, EmpName$, Salary
END SUB
```

2.

```
CALL HeadingModule
  :
SUB HeadingModule:
    LPRINT ...
      :
    LPRINT ...
RETURN
```

3.

```
CALL Calculate (X, Y, Z)
  :
SUB Calculate (A, B, C)
      :
END SUB
```

4.
```
CALL Add500 (Salary)
     .
     .
     .
SUB Add500 (NewSalary)
    NewSalary = Salary + 100
END SUB
```

§ 11.5.6 编程题

1. 编一程序计算利息、利率为9%至12%，年利息增加0.25%．时间为5年，本金从键盘输入，请用过程进行计算，公式为：总金额＝初始金额× （1+利率）年数．
2. 编程发工资．有两种方法，一是500元的基本薪加上所售额的15%；二是800元的基本薪加商品额的 10%；编程计算每个雇员的工资（给定人员、方法、商品售出额)．两种方法都要计算一次，并计算每种方法的平均工资，请使用过程．
3 运货至四个不同的地方，运往1区的价格为货价的6.5%．2 区为8%， 3 区为10%，4 区为 12%，输入顾客名、货物价格、信用卡号及区号．为每个区建立过程并打印输入的数据和运费．
4. 编程统计事故报告．输入的有人名、年龄、性别、保险与否．使用过程来统计汇总．

   a. 25岁以下司机的出事概率
   b. 男性司机的出事概率
   c. 未保险司机的出事概率

5. 编一程序让用户与计算机玩Til Tac Toe .

第十二章　自顶向下模块化程序设计中的函数、字符串处理及递归

本章要求:

用户应熟悉:

1. 编写程序时，函数的使用
2. 字符串处理
3. 局部变量，公共（全局）变量和静态变量.
4. 递归的概念.

§12.1　函数用作自顶向下、模块化结构

§12.1.1　函数简介

函数是一种程序结构，它和过程非常相似，函数和过程的主要区别是：过程仅是被用来执行一系列的操作，而函数却不仅如此，它还要给调用它的模块回一个值.

函数有两种类型：预先定义的函数和用户自己定义的函数(自定义函数). 预先定义的函数已经是 Turbo BASIC 语言的一部分.

例 1

Turbo BASIC 有一个平方根函数SQR，它返回一个数的平方根值. SQR 是一个预先定义的函数，它返回任何一个正的数字型变量或表达式的平方根，也就是说，当 SQR 函数被调用时，用来计算平方根的一组预先写好的指令集合被执行，这个函数象所有的函数一样，返回一个值，它可以用在赋值语句中，也可以作为表达式的一部分. 象 SQR 这样的函数能够用在赋值语句中:

```
a. INPUT X
   LET  Y=SQR (X)      '把 X 的平方根值赋给 Y
```

或者，它们也可以用在表达式中:

```
b. Z=SQR (36) +SQR (25)
   PRINT Z    '打印数值 11，它是由 (6+5) 得到的
```

最后，它们还可以用在 PRINT 语句中:

```
c. PRINT SQR(49)
```

例 2

```
INPUT W
LET   Z=INT (W)
```

预先定义的函数 INT 返回一个数的整数值，假设 W 输入为 15.65，这个赋值语句返回 15 赋给 Z. 因为 15 是 15.65 的整数部分. 也就是说，INT 对应着一组预先写好的令，称之为函数; 它可以脱离开程序主体被执行，这个函数返回变量的整数值. 这个变放在括号里，称之为参数. 这个整数值总是被截去，而不是四舍五入，如 IN(4.99)=4; INT (3.88) =3. 等等.

本章 § 12.1.3 节将介绍大量的预定义函数，注意 EOF 是一个预先定义的函数，在六和第七章已经用过. 在详细讨论其他预先定义的函数之前，让我们先考虑一个自定义数.

§ 12.1.2 自定义函数

自定义函数给主模块返回一个特定值，它可以如下编写:

例 1

```
'下面是程序的主体
LET  Y=FNF unction Name  (Value)
        :
        :
END

'这里是包含函数的模块
DEF  FNFunction Name (Value)
        :  }
        :  }      '这里是计算返回值的指令
        :  }
END  DEF
```

注意: 自定义函数总是以FN开始，子程序DEF ... ENDD DEF 计算出一个值被在 Y 中.

假设要在一个程序中计算从 1 到 N 的整数累加和，N 从 2 到更多的点取值；N 是不的，它取决于模块或函数调用什么时候被执行．可以在每一个点重复编程计算这个累加，但是如果用一个模块实现此目的，每当需要时就调用它将更少冗余．这个模块可以是个过程，也可以是一个函数，事实上，完成此任务用函数编程是最好的选择．因为这一列将被执行的操作需要返回一个特定的值，也就是整数累加和．

程序的主体将用一个函数调用语句来调用此函数．这个函数调用：

(1) 标识了将要执行的函数名．
(2) 传递参数给那个函数．

例子中，N 的值是变化的，以便于作为一个参数传递给那个函数．按如下格式调用该数．

```
SUM = FNsumOfIntegers (N)
```

其中：Sum　存放函数执行后的结果；
　　FN　调用函数；
　　SumOfIntegers　为函数名；
　　N　表示要传给该函数的参数．

如果这个函数总是被要求计算同样的累加和（如：从 1 到 100 的整数的量加和）．那就不必要给该函数传递任何参数．在这种情况下，FNSumOfIntegers 能被用来作为一函数调用，但没有参数．要求返回一个数值时，将省略这个参数．

函数自身被限定在 DEF ... ND DEF 结构内．假字 N 是用来计算从 1 到 N 的整数加和的参数，编写该函数如下：

```
DEF FNSumOfIntegers (N)
    SumInt = 0
    FOR Integer = 1 TO N
        SumInt = SumInt + Integer
    NEXT Integer
    FNSumOfIntegers = SumInt
END DEF
```

如果执行这个主程序并且 N 输入为 5，将打印下列结果：

```
The Sum of Integers 1-5 is 15
The Sum of Integers 1-3 is 6
```

可以在该程序中用 FN 函数调用在任何点执行一个函数．当一个函数调用被引用时，位于 DEF ... END DEF 内的语句将被执行．在这种情况下，Sum Of Integers 的值将被返回到调用模块，在 DEF FNSumOfIntegers ··· END DEF 中，可以用一条指令返回一个值，在上面的例子里，SumOfIntegers = sumInt 返回一个值给该函数．

例 2

该程序用函数计算工资:

```
'Program uses a function to calculate wages
CLS
INPUT Rate
INPUT Hours
Wages = FNTotalWages (Hours, Rate)
PRINT Wages
END
DEF FNTotalWages (Hours, Rate)
    IF  Hours > 40 THEN
        FNTotalWages = Rate * 40 + 1.5 * Rate * (Hours - 40)
    ELSE
        FNTotalWages = Rate * Hours
    END IF
END DEF
```

FNTotalWages 或者取 Rate × Hours 的结果，但当 Hours 超过 40 时，超时部分用 1.5 倍来计算. 在这里，DEF FNTotalWages … END DEF 又用一条指令 (FNTOalWages = eopression) 返回一个值. 一旦 FNTotalWages 函数定义完毕，它就可以用下列任何语句调用:

a. ' 注意这里的参数名字与函数中的不一样.
   Wages = FNTOtalWages (EmpHours, EmpRate)

b. ' 可以用常数直接作为参数
   Wages = FNTOtalWages (45, 5.00)

c. ' 可以打印一个函数调用的结果
   PRINT FNTOtalWages (45, 5.00)

d. '函数可以作为表达式的一部分
   BiWeeklyWages = FNTOtalWages (Hours1, Rate1) + _
   FNTOtalWages (Hours2, Rate2)

总而言之，不象过程调用那样，可以直接打印函数值；象上面的 C 那样，或者把它用在表达式中，而不必事先把它赋给一个变量. 函数也可以象上面的 d 那样用在表达式中；子程序却不能这样使用，也就是说，只有函数才能成为表达式的一部分，而过程和子程序都不行.

然而，可以用一个过程而不是函数编出上面的任何一个程序. 例如，在上面的 d 中，可以用下面的一个过程代替那个函数.

```
CALL  TotalWages (Hours1, Rate1, Wages1)
CALL  TotalWages (Hours2, Rate2, Wages2)
BiWeeklyWages = Wages1 + Wages
.
.
.
END
```

```
SUB TotalWages (Hours, Rate, Wages)
    IF  Hours > 40 THEN
        Wages = Rate * 40 + 1.5 * Rate * (Hours - 40)
    ELSE
        Wages = Rate * Hours
    END IF
END SUB
```

无论什么时候如果一个模块的目的是为了返回一个数值，用函数代替过程或子程序总是最好的选择。

用函数把一个值传递给某模块的通用格式是：

```
    Result = [FN name of function] (optional parameters)
    .
    .
    .
END
DEF [FN name of function] (optional parameters)
    .
    .     A value is calculated
    .
   [FN name of function]  = Calculated value
END DEF
```

例 3:

```
    Total = FNSumOfAAndB (A, B)
    .
    .
    .
END
DEF FNSumOfAAndB (A, B)
    FNSumOfAAndB = A + B
    .
    .
    .
END DEF
```

注意：可以把参数从调用模块传递给函数；函数可以调用其他子程序，并且其他子程序也可以调用函数。函数象其他子程序一样可以在程序的任何地方出现，但是建议用户在主模块的 END 语句之后编写函数，并且主函数应放在下边函数之前。

过程和函数之间的区别

注意函数总是返回特定值，然而过程可被用来执行一系列的操作，或者象函数一样，它能够返回一个值给主程序或调用程序。

函数调用

```
    PRINT "Enter Celsius Temperature"
    INPUT Temp
    PRINT "The Fahrenheit Equivalent Is "; FNCtoF(Temp)
    .
    .
    .
END
DEF FNCtoF(Temp)
    FNCtoF = (1.8 * Temp) + 32
END DEF
```

A value is returned by the function
This will be the Fahrenheit equivalent
of a Celsius temperature

上面的 FNCtoF (Temp) 是一个自定义函数，将返回一个值用来打印．该返回值是与输入的摄氏温度等价的华氏温度 (Fahrenheit) 值．

过程调用: **可以执行任何操作系列**

```
    INPUT A, B, C
    CALL TotalThem(A, B, C, Total)
    .
    .
    .
END
SUB TotalThem(A, B, C, Total)
.
.
.
END SUB
```

Any set of operations can be performed
here, including returning a value to the
calling module or printing a value

过程象函数一样能够返回一个值，它能执行任何指令序列．

单行函数

在介绍过程中，已说明了多行函数．单行函数有更为简单的结构:

DEF FN 标识符[(参数表)] = 表达式

例

```
    INPUT Principal, Rate
    PRINT "INTEREST FOR THE FIRST YEAR IS "; FNInterest(Principal, Rate)
    .
    .
    .
END
DEF FNInterest (Principal, Rate) = Principal * Rate
```

当自定义函数仅包含一行语句时，没有必要再使用 END 结束符；也就是说，DEF FunctionName = expression 作为单行函数没有必要后跟 END DEF。因为单行函数限制较多，故通常使用的是多行函数，就如前面描述的一样。

需要几行代码的自定义函数有一种格式，它必须包含一个 END DEF 范围终止符并且还可以包含一个 EXIT DEF，下面是自定义函数的正规格式：

```
DEF  FN 标识符[(参数表)]
         :
         :    } 语句组
         :
     [EXIT  DEF]
     [FN 标识符  =  表达式]
END  DEF
```

为了便于阅读，全部程序行缩进 DEF 和 END DEF 之间。

函数中的标识符，同子程序中的标号和过程名一样，是用户定义的并且必须少于或等于 31 个字符，而且也必须是唯一的。参数是一个或多个标识符的选择序列。它们之间用逗号隔开。代表当函数被调用时传递给函数的值。函数和过程一样，其参数表均被限制为最多 16 个参数。在函数里，表达式定义了一个"处理"，这个"处理"被用来计算该函数将要返回的值。

EXIT DEF 是一个可在计算执行前用来提前退出函数的选择，无论什么时候参数表中发现了错误，EXIT DEF 就被用来退出这个函数结构。因此，EXIT DEF 和 EXITSUB 是相似的。EXIT SUB 用来提前退出过程。但务请注意，在子程序中不允许使用 EXIT。

程序行：

```
DEF  FN 标识符  =  表达式
```

把值赋给函数名。虽然这是一个选择项（它被括在括号里），但总是包含它；如果没有这一行，就不会把值返回到调用模块。

END DEF 终止一个多行函数并且把控制返回到调用该函数的语句。

例

下面这个函数用来计算一个整数的阶乘，该整数从 1 到 10 取值。阶乘按如下方法计算：$3! = 3\times2\times1$；$5! = 5\times4\times3\times2\times1 = 120$；$n! = n\times(n-1)\times\cdots\times1$。

```
DEF FNFactorial#(X%)                'X% is an integer variable
    IF X% < 1 OR X% > 10 THEN       'Error in X%
        FNFactorial# = 0
        EXIT DEF
    END IF
    Total# = 1
```

```
        FOR I% = 2 TO X%
            Total# = Total# * I%
        NEXT I%
        FNFactorial# = Total#
    END DEF
```

这个自定义函数用来计算阶乘,它可按如下方法调用:

```
PRINT "Enter any integer from 1 TO 10"
INPUT X%
PRINT "The Factorial Is "; FNFactorial#(X%)
```

在"FN 标识符"中的#和 Total 后的#，表示所使用的是双精度数．双精度数比单精度数更长也更精确．

有一个不成文的约定，那就是在 PRINT 语句中只打印变量和（或）常数，而不打印表达式；这是一个很好的习惯．按此约定，可以把上面的程序改造如下:

```
PRINT "Enter any integer from 1 to 100"
INPUT X%
Y# = FNFactorial#(X%)
PRINT "The Factorial is "; Y#
```

函数不改变在调用模块中定义的参数内容:

```
X = 55
Y = FNFunction1(X)
PRINT X ←——— X prints as 55 even if X is changed in Function1
```

其中：即使函数Function1改变了X，打印的也是：15

也就是说，函数不同于过程，总是使用按值传递方法传递参数. 这就意味着函数根本就不能使用按引用传递参数的方法．这是因为函数仅给调用模块返回一个值．它不能用来改变传递给它的值的内容．但对于过程而言，当它使用按引用传递参数方法时可以实现该功能．

考虑如下函数:

```
DEF Function1 (Amt1)          FNFunction1 (Amt1)
    .                         FNFunction1 (3)
    .                         FNFunction1 (Total1)
    .                         FNFunction1 (Amt2 - Amt3)
END DEF
```

但是应该注意：即使 Amt′在该函数之内被改变．这对于该函数调用中所限定的变量

也没有任何影响（这些变量包括上面的 Amt，Total1，Amt2ak Amt3）。

必须保证在函数调用（FN）中使用的标识符和相应的函数定义（DEF）中的标识符完全一样：

非法（错误）编程：Function A 和 Function A1 不是相同的标识符

```
FNFunctionA (2, 3)
     :
DEF FNFunctionA1 (A, B)
```

当函数调用（FN）时其参数表中的自变量或值的个数必须和相应的函数定义(DEF)中的参数表一致。下面的调用将导致参数不匹配错误。

错误编程：参数不匹配错误

```
FNFunctionX(3, 4, 5) ←— 3 parameters defined here
:
DEF FunctionX(A, B) ←— Only 2 parameters defined here
```

函数能返回数值型值或字符串值。函数的类型，同变量的类型一样，取决于它的名字，因此返回字符串值的函数其名字必须以 $ 结束。例如，假设编写：

X $ = FNidentifier (parameger list)

则标识符（idmtifior）必须以 $ 结束，因为它要返回一个字符串类型的值。

例

FNFunction1 (X)	←	带一个单精度数值型自变量X并且返回一个数值型的值。
FNFunction2 $ (Y)	←	带一个单精度数值型自变量Y并且返回一个字符串类型的值。
FNFunction3% (Z%)	←	带一个整数类型的自变量Z%并且返回一个整数类型的值。
FNFunction4# (W%)	←	带一个整数类型的自变量W%并且返回一个双

精度类型的值.

如果试图用字符串类型的函数返回一个数值型的值，或任何相似类型的变量将产生语法错误. 例如，下面的调用将产生语法错误:

X$ 应该是 X，或 FunctionQ 应该是 FunctionQ$
X$ = FNFunctionQ (Y)

产生的错误是:

```
Error 13: Type Dismatch
```

自我测验(一)

1. 到目前为止本段所讨论的子程序三种类型分别是________、________、________和________.
2. 函数和过程的区别在于________.
3. 调用函数时，在函数名或标识符之前使用________.
4. (T或F) 函数所带的参数可有可无.
5. (T或F) 单行函数没有必要带END语句.
6. (T或F) 函数能调用其他的函数.
7. (T或F) EXIT DEF 可用来提前离开 (退出) 函数
8. (T或F) 多行函数必须以DEF开始，以DEF结束.

考虑下面这段程序，并回答题9～14:

```
    LET Amt1 = 15
    LET Amt2 = 10
    SUM = FNSumIt(Amt1, Amt2)
    PRINT "The Sum is "; Sum
    .
    .
    .
END
DEF FNSumIt(Amt1, Amt2) = Amt1 + Amt2
```

9. (T 或 F) 上面的函数使用单行函数正确地实现了 Amt1 加 Amt2
10. (T或F) 上面的函数可在程序中不同的地方被调用，并且给Amt1和Amt2 赋以不同的参数值.
11. (T或F) 该函数也可编写如下:

```
DEF FNSumIt(Amt1, Amt2)
    FNSumIt = Amt1 + Amt2
END DEF
```

12. 重新编写FNSumIt 以便它能返回Amut1+Amut2的平方值.
13. (T或F) FNSumIt 也可用来改变Amut1t Amut2的值.
14. 编写一个函数，返回三个数的平均值，并且包含该函数调用的例子.
15. 编写一个函数，返回从Z到N之间偶数的累加和，N值从主程序传送到该函数.

§ 12.1.3 预定义函数

下面是一张预定义的 Turbo BASIC 函数表，它们分为:

(1) 对数值类型的变量进行操作的函数，它们的大多数返回数值类型的值.

(2) 对字符串类型的变量进行操作的函数，它们中的大多数返回字符串类型的值.

预定义的数值类型函数	
ABS	求绝对值
ASC	求字符串中第一个字符的 ASCII 码值
ATN	求反正切值
CDBL	将数据转换为双精度浮点数.
CEIL	返回一个大于或等于其自变量的最小整数.
CINT	对数据进行舍入取整
CLNG	将数值转换为长整型数
COS	返回自变量的余弦值
CSNG	将数值转换为等价的单精度值
CVI	将从随机文件中读取的字符串型数据转换为数值型.
CVMD	将从 Microsoft 格式的随机文件中读取的字符串型数据转换为 IEEE 格式的浮点数
DECR	设置或获取系统日期
DEFINT / DEFLNG / DEFSNG / DEFDBL / DEFSTR	定义变量的缺省类型
EXP	指数运算: e^x
EXP2	指数运算: 2^x
EXP10	指数运算: 10^x
FIX	截去小数部分，取整.
INCR	增大变量

INT	将数值取整
LOG	计算自然对数(以 e 为底)
LOG2	计算对数(以 2 为底)
LOG10	计算对数(以 10 为底)
MKI$ / MKL$ / MKS$ / MKD$	将数据转换为要输入到随机文件中的字符串
MKMS$ / MKMD$	将数据转换成 Microsoft 格式的字符串（用于随机文件的输出)
RANDOMIZE	随机数产生器的种子
RND	产生随机数
SGN	符号（判断）函数
SIN	返回正弦值
SQR	返回数值的平方根
TAN	返回正切值

例：严格的数据类型的函数

1. 计算625的平方根，把结果放在Y内
   Y=SQR (625)

2. 计算变量Sum的整型值并把它放在Sum2内
   Sum2=INT (Sum)

3. 计算一个0到1之间的随机数并把它放在Y内
   Y=RND

4. 计算一个1到n（包含n）之间的随机数
   RandomNo%=INT (RNDkn) +1

当描述预先定义的函数时，规定 x 为数值型变量（如 ABS (x)，INT (x))，但是，预先定义的函数也同样可以对数值型表达式进行操作.

附加例:

```
X = SQR (Z*Z)        —' 返回 Z*Z 的平方根
Y = ABS (X-15)       —' 返回 x-15 的绝对值.
```

其他预先定义的函数包括:

字符串操作	
BIN $	返回一个数值的二进制形式字符串
CHR $	返回 ASCII 码对应的字符
COMMAND $	在 DOS 状态下启动程序时，获取命令行中的其它信息.
GET $	从二进制文件中读取数据
HEX $	将十进制数转换为相应的十六进制格式字符串
INSTR	字符串查找
LCASE $	将字符串中的字母全部变为小写
LEFT $	获取字符串最左边的 n 个字符
LEN $	求字符串的长度
MID $	替换字符串／自字符串中提取子字符串
OCT $	返回数值的八进制形式字符串
RIGHT $	从字符串的右边开始提起子字符串
SPACE $	返回指定数目的空格字符串
STR $	将数值转化为字符串
STRING $	返加若干个相同字符给成的字符串.
TIMER $	设置系统时间
TIMER $	返回系统时间
UCASE $	将字符串中字母全部变成大写字母
VAL	将字符串转换为数值

带有字符串参数的函数大都返回一个字符串类型的值；不过，函数 LEN、INSTR 和 VAL 除外，它们都返回数值类型的值.

预先定义的字符串 LEFT $ 和 RIGHT $

LEFT $ 和 RIGHT $ 的格式如下：

$$S\$ = \left\{ \begin{matrix} LEFT\$ \\ RIGHT\$ \end{matrix} \right\} (\text{字符串表达式, N})$$

其中：N 为返回的字符数目.

字符串函数调用举例

1. S$ =LEFT$ ("JOHN KENNEDY", 4)
   S$将包含"JOHN".

2. N$ =RIGHT$ ("HOHN SMITH", 5)
   N$将包含"SMITH".

3. M$ =MID$ ("Happy Bithday" 7,8)
   M$将包含"Bithday".

4. LENGTH=LEN (EXTRAORDINARY")
   LENGTH将包含13.

5. LOWER$ =LCASE$ ("Robert louis stevenson")
   LOWER$将包含"robert louis stevenson"

6. Z=VAL (.543.92")
   Z将包含543.92

7. 把"grape"的等价大写形式放在变量Fruit$
   Fruit$ =UCASE$ ("grape")

8. 输入一个字符串变量，执行一个循环直到字符串变量="LAST"，而且可以保证即使"Last"以小字母形式输入，这个循环也将终止．该程序框架如下:

```
INPUT String1$
String2$ = UCASE$(String1$)
DO  WHILE String2$ <> "LAST"
     .
     .
     .
     INPUT String1$
     String2$ = UCASE$(String1$)
LOOP
END
```

§ 12.2　使用预先定义的函数进行字符串处理

§ 12.2.1　输入字符串数据的方法

§ 12.2.1.1　用 INPUT 语句提示用户

至此，已经重点讨论了用于输入数据的基本的 INPUT 语句：

```
INPUT variable1, variable2 ...
```

其中的每一个变量既可以是数值型也可以是字符串类型。第二章指出过输入语句还可包含提示：

例:

```
INPUT "Enter salary"; salary
```

当程序执行时，这条指令将在 Run 窗口中给出如下显示：

Enter Salary?　　←　键盘输入将在此行里显示

当数据输入时，它出现在提示行上，这对于很多应用而具有特别的意义，注意? 作为提示符自动跟在显示的信息之后，就同不带显示信息的 INPUT 语句执行时一样。

程序员或系统分析员应该把用户和计算机之间的接口设计的尽可能的清楚。到目前为止，已讨论过的两种可能的选择是：

编　辑	运　行
1. PRINT "Enter Salary	Enter Salary
INPUT Salary	?_　←　输入的数值在此显示
2. INPUT "Enter Salary"; Salary	Enter Salary ?_　←　输入数值在此出现

注意：在例2中，INPUT语句中的分号（;）把文字常数和变量分隔开来；如果使用逗号代替，结果就不同：

编　辑	运　行
3. INPUT "Enter Salary", Salary	Enter Salary ?_　←　输入数值在此出现(没有?)

因此，当用逗号而不是分号来分隔打印信息和将要输入的数值时，没有"?"出现。使用这种格式时，必须保证在包含在引号中的信息的末尾留一空格。

总而言之，无论何时 INPUT 指令都会为需要的输入进行提示，被键入的数据总是出现在和提示信息相同的行上。

指　　令	提示信息后有问号吗?	输入数据在屏幕的显示位置
1. PRINT "Enter Amount" INPUT Amount	有——在第 2 行	第 2 行
2. INPUT "Enter Amount"; Amount	有——在第 1 行	第 1 行
3.INPUT "Enter Amount";	没有	第 1 行

考虑图 12.1 所述情况:

EDIT 窗口

```
INPUT "enter salary ";salary
PRINT "salary is";salary
PRINT
PRINT "enter salary"
INPUT salary
PRINT "salary is ";salary
PRINT
INPUT "enter salary ",salary
PRINT "salary is ";salary
```

RUN 窗口

```
enter salary ? 10000
salary is 10000

enter salary
? 20000
salary is 20000

enter salary 30000
salary is  30000
```

图 12.1 交互显示的几种类型

§ 12.2.1.1 行输入（LINE INPUT）语句

假定用户想把数据输入到字符串变量 Address $ 中，下面的尝试将产生错误的结果。

编　辑	运　行
PRINT "Enter Address"	Enter Address
INPUT Address $	?100 Main St., NY, NY 10036

如果敲入 Addres $ 的内容，用户将会发现它仅包含“100 Main St.”，而没有输入地址的剩余部分，这是因为跟在 100 Main st. 后的程序被计算机解释为该变量的数据到此为止，再也没有其他的数据.

LINE INPUT 语句使用户能够把从键盘输入的一行数据读到字符串变量中，而忽略了作为定界符的逗号.

LINE INPUT 指令的格式如下:

```
LINE   INPUT [;] [提示字符串; ]字符串变量
```

在 LINE INPUT 语句中可以使用提示字符串，字符中变量将被装入存贮器，在该变量中用户可以任意使用逗号.

考虑下述情况:

Edit

```
   C:LINEINP.BAS Line 7 Col 7
CLS
INPUT Address1$
PRINT "Incomplete address is: "
PRINT Address1$
LINE INPUT Address2$
PRINT "Complete address is: "
PRINT Address2$
```

Run

```
? 100 Main St., NY, NY 10036
Incomplete address is:
100 Main St.
100 Main St., NY, NY 10036
Complete address is:
100 Main St., NY, NY 10036
```

注意: LINE INPUT 指令也不产生作为提示符显示的“?”.

LINE INPUT 指令常用来读入包含逗号的数据. 在此，逗号是有效字符; 也可以用 LINE INPUT#语句从顺序文件中把一行数据读入到一个字符串变量，而忽略定界符.

总而言之，有很多输入数据的方法，用户可以根据具体的应用场合进行选择.

§ 12.2.1.3 INPUT $ 函数

标准的 INPUT 指令允许输入任何数目的字符，只有当用户按回车键时输入传输才结束.

假定想限制输入字符的数目，也就是说，想把输入的 Salary 准确地限定为 5 个数字或把输入的 Transaction 数值精确地限定为 4 个数字; 此时，可以用函数 INPUT $ (n) 实现该目的.

函数 INPUT $ 的基本格式如下:

```
字符串变量 = INPUT$ (n)
```

在此，n 指定了将要输入字符的准确数目.

例:

```
PRINT  "Enter transaction # as 4 digits"
TransNo $  =  INPUT $ (4)
```

注意: 如果使用键盘输入，输入的数据又会象其他的输入方式那样在屏幕上显示，因此当输入诸如工资或关键字（口令）等秘密数据时，函数 INPUT $ (n) 是非常有用的.

如果输入的字符个数小于规定的数目 n，那么计算机将继续等待输入直到凑够几个字符为止；如果输入的字符数目多于规定的数目，那么多余的字符将被忽略，因此，当使用函数 INPUT $ 输入一个事项，数据然后用 PRINT 语句打印在屏幕上时将产生如下交互信息:

Edit

```
            IN$.BAS     Line 4    Col 22    I
CLS
PRINT "enter transaction # as 4 digits"
TransNo$ = INPUT$(4)
PRINT "trans no is "; TransNo$
```

Run

```
enter transaction # as 4 digits
trans no is 3456
```

注意: 函数INPUT $ 假定输入是字符串形式；如果想对它执行算术操作，则首先必须把它转变成数值型的变量.

函数 INPUT $ 的完整格式允许从文件中输入数据:

字符串变量 = INPUT $ (#) [文件号]

§ 12.2.1.4 函数 INKEY $

函数 INKEY $ 同函数 INPUT $ 一样，读入数据时也不在屏幕上显示输入的字符，然而它不能用来从文件中输入数据. 它最常用于判断输入的数据类型. 而不是实际的数据本身.

函数 INKEY $ 的格式如下:

```
字符串变量 = INKEY $
```

当该指令执行时，字符串变量中将包含 0，1，或 2 这三种字符之一，至于取何值；这取决于已经键入的数据，这个字符串指键盘缓冲区的状态。

字符串变量	含　义
0（空字符串）	没有击键（也就是说，没有数据输入）
1	键入一个 ASCII 字符
2	键入了一个非 ASCII 字符

为了判断是否击键，需要检查字符串变量的长度（LEN），如果 LEN 等于 0，则没有击键。

和其他的输入函数不同，INKEY $ 并不等待键盘输入数据，而是检查键盘缓冲区，看在此之前是否有键盘输入。

函数 INKEY $ 的一个用途是：只需要按任意一键；比如回车键，即可中止一个循环或过程的执行。

回车键的 ASCII 代码是 13，可以指导用户建立一个字符串，完成时按回车键结束。

Edit

```
PRINT "Enter a character at a time"
PRINT "Press the Enter key when done"
DO  WHILE CHar$ <> CHR$(13)
    Char$ = INKEY$
    EnteredString$ = EnteredString$ + Char$
LOOP
PRINT "String is "; EnteredString$
END
```

Run

```
Enter a character at a time
Press the Enter key when done
String is abcde12
```

注意，用函数 INKEY $ 输入的数据在屏幕上不回显。

§ 12.2.2 举例: 如何用字符串函数对字符串进行操作

假定输入"Bruce Springsteen"作为对 INPUT 语句的响应，请考虑下面的情况:

```
CLS
INPUT "Name: ";NameIn$
A$ = LEFT$ (NameIn$, 5)
B$ = RIGHT$ (NameIn$, 11)
C$ = MID$ (NameIn$, 6, 2)
D$ = LEFT$ (NameIn$, 1) + MID$ (NameIn$, 7, 1)
E$ = D$ + " " + MID$ (NameIn$, 7, 3)
        + RIGHT$ (NameIn$, 8)
F = LEN (NameIn$)
PRINT "A$ = ";A$
PRINT "B$ = ";B$
PRINT "C$ = ";C$
PRINT "D$ = ";D$
PRINT "E$ = ";E$
PRINT "F = ";F
END
```

RUN 窗口将显示如下:

```
Name: ?Bruce  Springsteen
A$—=  Bruce
B$—=  Springsteen
C$—=  S
D$—=  BS
E$—=  BS  Springsteen
F—=  17
```

A$ =LEFT$ (NameIn$, 5) 把 NameIn$ 中最左面的五个字符放在 A$ 内. 因此, Bruce 被送到 A$ 中. 同理, B$ =RIGHT$ (NameIn$, 11) 把 NameIn

中最右边的 11 个字符 Springsteen 放在 B$ 中, C$ =MZD$ (NameIn$, 6, 2) 把第六和第七个字符放入 C$ 中; 而 D$ 和 E$ 则把字符串函数联接起来, F=LEN (NameIn$) 把该字符串变量的长度放在 F 内.

自我测验(二)

考虑如下字符串:

String1$ = "School is almost over and I'm glad"

1. 请说明下面程序的执行结果.

```
CLS
String1$ = "School is almost over and I'm glad"
A$ = LEFT$ (String1$, 21)
B$ = RIGHT$ (String1$, 4)
C$ = LEFT$ (String1$, 0)
D$ = RIGHT$ (String1$, 8)
E$ = D$ + " it " + MID$ (String1$,8, 14)
F = LEN (String1$)
PRINT "A$ = ";A$
PRINT "B$ = ";B$
PRINT "C$ = ";C$
PRINT "D$ = ";D$
PRINT "E$ = ";E$
PRINT "F = ";F
END
```

2. 编写一个程序，输入电话号码，并将它定义为Number $；它包括三部分：地区号码（area code），交换机号码（exchange）和最后四位数字（例如：516-560-6716）．按如下格式打印：

   Area Code =
   Exchange =
   Last 4 digit =

3. 为什么Number $ 必须定义为字符串变量?

下面考虑几个处理字符串的程序样例:

例 1

使用 MID $ 函数把名字从"Last Name, First Name" 格式变为："First Name Last Name"格式。

假定输入名字时先输入名字（Last name），然后输入姓（first name），中间用逗号分隔（逗号后面不留空格）。希望把姓名打印出来时，姓（first name）在前，名字（Last name）在后，中间由逗号分开．使用函数 MID $ 找到把名字和姓分开的逗号，然后分别处理每一部分．为此，进行如下操作：

1. 把名字，姓（带逗号）输入到变量FullName $ 中。
2. 在变量FullName $ 中找到逗号出现的位置，然后把该位置贮存在变量LocationCtr 中。
3. 姓（first name）贮存在FirstName $ 中，它是FullName $ 的一个字符，从 LocationCtr+1 开始，到 FullName $ 的最后一个字符为止。
4. 名字（Last name）贮存在LastName $ 中，它也是FullName $ 的一个字串，从 FullName $ 的第一个字符开始，到第 LocationCtr − 1 个字符为止。
5. 先打印FirstName $，然后打印LastName $ 中间用逗号隔开。
6. 重复整个过程直到不再输入。

带有注释的整个程序参见图 12.2. 这个程序将打印姓（First Name），名字（Last Name），中间没有逗号而用空格分开.

EDIT 窗口

```
'*************************************************************
'This program illustrates string processing
' By entering last name, first and printing first name, last
'*************************************************************
CLS
PRINT "Enter Last Name first then a comma then First Name"
MoreInput$ = "Y"
DO WHILE MoreInput$ = "Y" or MoreInput$ = "y"
        LINE INPUT FullName$
        LocationCtr = 1
'*********************************************************
'The following DO WHILE finds the position of the comma
'*********************************************************
        DO WHILE MID$ (FullName$, LocationCtr,1) <> ","
                LocationCtr = LocationCtr + 1
        LOOP
'*************************************************************
'FirstName$ is assigned the value in FullName$ from the position
'after the comma until the end of the FullName$ variable
'*************************************************************
        FirstName$ = MID$ (FullName$, LocationCtr+1, LEN (FullName$))
'*************************************************************
'LastName$ is assigned the value of FullName$ from position 1 until
'the position preceding the comma
'*************************************************************
        LastName$ = MID$ (FullName$, 1, LocationCtr - 1)
        PRINT FirstName$; " " ; LastName$
        PRINT "More Input? (Y/N)"
        INPUT MoreInput$
LOOP
PRINT "End of Job"
END
```

RUN 窗口

```
Enter Last Name first then a comma then First Name
Newman,Paul
Paul Newman
More Input? (Y/N)
? Y
Evans,Linda
Linda Evans
More Input? (Y/N)
? Y
```

```
Mouse,Mickey
Mickey Mouse
More Input? (Y/N)
? N
End of Job
```

图 12.2　例 1 的程序：字符串函数

例 2

同例 1 一样，不过在此用函数 INSTR 在字符串 FullName $ 中查找逗号．

INSTR 的格式如下：

INSTR(n，目标字符串，模板字符串)

从位置 n 开始搜查目标字符串；从中查找模板字符串，可以从位置 1 开始在 FullName $ 中查找“，”；“，”是模板字符串，INSTR 将返回目标字符串中模式字符串的位置．因此 INSTR 将指明 FullName $ 中逗号所在的位置．

INSTR 能够取代图 12.2 中的 DO WHILE … LOOP 结构体．方法如下：

```
LocationCtr = INSTR (1,FULLName $ ",")
```

例 3

输入一行正文然后找出：

(a)　该正文中元音字母出现的个数．

(b)　该正文中辅音字母和其他字符出现的个数．

使用函数 LEN 计算该正文的长度，用函数 INSTR 查找元音字母出现的个数，用函数 MID $ 从输入字符串中每次取出一个字符，然后给相应的计数器加 1．

图 12.3 给出了整个程序．FOR … NEXT 循环将正文中的每一个字符用作模板字符串，该字符从目标字符串的第一个位置开始比较．目标字符串由大写的全部字符元音字母“AEIOU”组成．如果模板字符串中的字符等于其中一个元音字母，那么 1 将加到 NoOfVowels；否则，NoOfOtherCharacters 加 1．因此，使用下述结构：

```
FOR  I% = 1  TO  n
        IF  INSTR(1, "AEIOU", MID $ (TextLine $ , I%, I)  THEN
                :
NEXT
```

IF 语句把 Textline $ 中的每一个字符都同元音 AEIOU 进行比较．

EDIT 窗口

```
'**********************************************************
' This program enters a line of text and counts the # of
' occurrences of vowels and other characters
' All text is assumed to be entered in upper case
'**********************************************************
CLS
PRINT "Enter a line of text using upper case letters"
MoreInput$ = "Y"
DO WHILE MoreInput$ = "Y"
        LINE INPUT TextLine$
        LengthOfText = LEN (TextLine$)
        NoOfVowels = 0
        NoOfOtherCharacters = 0
        FOR I% = 1 to LengthOfText
                IF INSTR (1, "AEIOU", MID$ (TextLine$, I%, 1)) THEN
                        Let NoOfVowels = NoOfVowels + 1
                ELSE
                        Let NoOfOtherCharacters = NoOfOtherCharacters + 1
                END IF
        NEXT I%
        PRINT "No Of Vowels is ";NoOfVowels
        PRINT "No Of Other Characters is ";NoOfOtherCharacters
        PRINT "MORE INPUT (Y/N)?"
        INPUT MoreInput$
LOOP
PRINT "END OF JOB"
END
```

RUN 窗口

```
Enter a line of text using upper case letters
FOUR SCORE AND SEVEN YEARS AGO
No Of Vowels is  11
No Of Other Characters is  19
MORE INPUT (Y/N)?
? Y
OUR FATHERS BROUGHT FORTH
No Of Vowels is  7
No Of Other Characters is  18
MORE INPUT (Y/N)?
? N
END OF JOB
```

图 12.3　例 3 的程序：字符串函数

例4 在一正文中查找字母"A"出现的次数，假定该正文以大写形式输入，打印A出现的次数。其程序参见图 12.4:

EDIT 窗口

```
'**************************************************************
'This program prints the # of occurrences of the letter A in a text
'**************************************************************
CLS
PRINT "Enter a line of text using upper case letters"
MoreInput$ = "Y"
DO WHILE MoreInput$ = "Y"
        LINE INPUT TextLine$
        LocationOfA = INSTR (1, TextLine$, "A")
        IF LocationOfA = 0 THEN
                Exit IF
        ELSE
                NoOfAs = 0
                DO WHILE LocationOfA > 0
                        LocationOfA = INSTR (LocationOfA + 1, TextLine$, "A")
                        NoOfAs = NoOfAs + 1
                LOOP
        END IF
        PRINT "No of A's in text is ";NoOfAs
        PRINT "MORE INPUT (Y/N)?"
        INPUT MoreInput$
LOOP
PRINT "END OF JOB"
END
```

RUN 窗口

```
Enter a line of text using upper case letters
THERE IS ONE A ON THIS LINE
No of A's in text is  1
MORE INPUT (Y/N)?
? Y
THERE ARE TWO A'S ON THIS LINE
No of A's in text is  2
MORE INPUT (Y/N)?
? Y
AT THIS LINE THERE ARE THREE A'S
No of A's in text is  3
MORE INPUT (Y/N)?
? N
END OF JOB
```

图 12.4 例 4 的程序：字符串函数

在该程序中，首先使用函数 INSTR 判断是否有"A"出现。如果没有，将退出循环；如果在正文中至少有一个"A"，则判定它的位置，给 NoOfAs 加 1，然后从最后找到"A"的位置之后继续处理字符串。TextLine $ 。该处理过程将一直持续到在此正文字符串中不再有"A"为止。

例 5: 不使用函数 UCASE $ 把一正文变成大写形式。

大写字母的ASCII代码比相应的小写字母的ASCII 代码小32。也就是说，"A"的 ASCII 代码是 65，而"a"的代 ASCII 的代码是 98。"B"的 ASCII 代码是 66，而"b"的 ASCII 的代码是 98，等等。可以利用该规律把小写字母变成大写字母。

整个程序的说明参见图12.5:

EDIT 窗口

```
CLS
PRINT "Enter a line of text using either upper or lower case letters"
MoreInput$ = "y"
DO WHILE MoreInput$ = "Y" or MoreInput$ ="y"
        LINE INPUT TextLine$
        LengthOfText = LEN (TextLine$)
        UpperCaseText$ = "" 'We begin with a null string in this variable
        FOR I% = 1 TO LengthOfText
                CharacterOfText$ = MID$ (TextLine$, I%, 1)
                IF CharacterOfText$ >= "a" AND CharacterOfText$ <= "z" THEN
                        CharacterOfText$ = CHR$ (ASC (CharacterOfText$) - 32)
                END IF
                UpperCaseText$ = UpperCaseText$ + CharacterOfText$
        NEXT I%
        PRINT
        PRINT "UPPER CASE VERSION OF THE ENTERED TEXT IS:"
        PRINT
        PRINT UpperCaseText$
        PRINT
        PRINT "More Input (Y/N)?"
        INPUT MoreInput$
LOOP
END
```

RUN 窗口

```
Enter a line of text using either upper or lower case letters
THis tEXt HAS uppER and LoWer CAseS AlL MiXED Up

UPPER CASE VERSION OF THE ENTERED TEXT IS:

THIS TEXT HAS UPPER AND LOWER CASES ALL MIXED UP
```

```
More Input (Y/N)?
? Y
THIS TEXT IS ALREADY IN UPPER CASE

UPPER CASE VERSION OF THE ENTERED TEXT IS:

THIS TEXT IS ALREADY IN UPPER CASE

More Input (Y/N)?
? y
this text is in lower case when entered

UPPER CASE VERSION OF THE ENTERED TEXT IS:

THIS TEXT IS IN LOWER CASE WHEN ENTERED

More Input (Y/N)?
? n
```

图 12.5 例 5 的程序: 字符串函数

自我测验(三)

判断下面各式的结果:

1. P = INT (-53.23)
2. Q = INT (2 * 3.18 + .9)
3. R = SQR (.16)
4. S = INT (SQR (.25))
5. T = ABS (-45.07)
6. A = CEIL(345.65)
7. B = CINT(47.4)
8. C = CINT(46.9)
9. D = EXP2(5)
10. E = EXP10(5)
11. F = FIX(72.95)
12. G = INT(72.95)
13. H = SGN(-23.48)
14. I = SGN(23.48)
15. K$ = "supercomputers"
    J = INSTR(K$, "computer")
16. L$ = LCASE$(Turbo Basic')
17. K$ = "Kemeny and Kurtz"
    M = LEN(K$)
18. O$ = "microcomputer"
    N$ = LEFT$(O$, 5)

19. P$ = STRING$(10, "surprise")
20. Q$ = UCASE$("Turbo Basic")

§12.3 在过程和函数中说明变量

Turbo Basic 不同于其他的程序设计语言，不需要对程序中的变量进行显式说明。当变量在任何可执行语句中使用时它们被隐式说明。

也就是说，指令 A=A+1 将建立一个变量 A。如果 A 事先没有定义的话，在一些语言里，必须在说明段建立程序中要使用的全部变量。但是 Turbo Basic 显得更灵活。然而这种灵活性可能导致错误。如果在一条指令中用 Amtil 访问一个变量，而另一条指令中用 Amt2 引用它，不会得到任何警告和错误的信息。编译程序将把它们作为两个不同的变量进行处理。但是反过来，如果有一个说明段（比如在其他的语言里）并且 Amt2 不包含在该段中，那么编译程序能很容易地把任何对 Amt2 的引用标识为一个错误。

在 Turbo BASIC 中，可以通过把过程或函数中的变量显式地说明为局部变量或共享变量来消除一些潜在的错误。局部变量是指仅在一个过程或函数中使用的变量，共享变量是指既能在指定的过程或函数中使用也能在其他模块中使用的变量。

§12.3.1 局部变量

考虑下面的程序：

```
'Main program goes here
T = 0
CALL Proc1(X)
PRINT "T is : "; T
.
.
.
END
'Subprograms go here
SUB Proc1(X)
    .
    .
    .
    T = 5
    .
    .
    .
END SUB
```

在主程序中定义了一个变量 T 并初始化为 0；在过程中也有一个 T，赋值为 5。问题是：过程中变量 T 的变化是否会影响到主程序中的变量 T。注意 T 不是参数，因此，它不会自动地从调用模块传递到被调用模块，当然，也不会从被调用模块自动地返回到调用模块。

在过程中说明了任何变量其缺省性都是局部变量，比如象上面的 T；也就是说，它们

不会影响该过程之外的任何对 T 的引用．事实上，尽管主程序中的 T 和过程中的 T 有相同的名字，它们都被认为是各自独立的变量．因此，在过程中的变量都被隐式地说明为局部变量，这意味着在上面的例子中 CALL 之后将打印"T is 0"，事实上，在被调用模块中 T 赋值 5 对调用模块中 T 的值没有任何影响．

如上所述，即使过程中的变量其缺省属性是局部变量，然而最好还是在过程中把这些变量显式地定义为局部变量．下一部分将介绍如何忽视这种缺省，并且把过程中的变量规定为共享变量．

但应注意的是：函数中的变量，同过程中的变量不同．其缺省属性是共享变量，这意味着如果函数改变了同名变量的值，那么在主程序中的变量的值也将改变．

例：

```
'Main program goes here
T = 0
Y = FNFcn1(X)
.
.
.
PRINT "T is : "; T
.
.
.
END
'Subprograms go here
DEF FNFcn1(X)
     .
     .
     .
     T = 5
     .
     .
     .
END DEF
```

在这个例子中，因为使用变量 T 的子程序或模块是函数而不是过程，主程序中的变量 T 将随被调用模块中 T 的变化而缺省地改变．因此在这个例子中函数调用之后将打印"T is: 5"．

因此，函数隐式地把变量说明为共享变量．为了忽视这种缺省，必须把它们说明为局部变量．过程把变量缺省为局部变量，这种缺省可通过把变量规定为共享变量而被忽略．现在，说明如何将这些缺省属性忽略．

在过程或函数内局部变量可定义如下：

```
SUB PROC1 (x, y, z)                 DEF FNFunction1(x, y, z)
   +---------------------+             +---------------------+
   | LOCAL variable list |             | LOCAL variable list |
   +---------------------+             +---------------------+
   .                                   .
   .                                   .
   .                                   .
END SUB                             END DEF
```

下面是过程或函数中说明局部变量表的例子：

```
SUB ...                         DEF...
    LOCAL Amt1, EmpName$            LOCAL A$, B, C$, D
    .                               .
    .                               .
    .                               .
END SUB                         END DEF
```

如果想把变量说明为局部变量（LOCAL），那么 LOCAL 语句必须是 SUB 或 DEF 之后的第一条语句，而且应该在任何操作执行之前说明。

在子程序中把变量显式地定义为局部变量可以保证：在函数或过程对变量执行的任何操作将不会影响主程序中任何其他的变量，即使其他变量碰巧具有相同的名字，虽然在过程中缺省属性就是局部变量，但是进行显式说明总是最好的选择。

注意：子程序和函数不一样，它不能把变量定义为局部变量，这使得子程序比过程或函数更容易出错误。也就是说，在一个非常长的程序中，如果在主模块和子程序中或在函数中没有把这些变量定义为 LOCAL，而使用了同样名字的变量，将导致大错误。

如果无意中在主程序和子程序中使用相同名字的变量，就会发生错误。参见下面的例子：

例：

```
'Main program goes here
EmployeeCount = 10
GOSUB CalcChecks
N = 1
GOSUB PrintChecks
.
.
.
END
CalcChecks:
    GOSUB CalcDeductions
    .
    .
    .
RETURN

CalcDeductions:
    FOR N = 1 TO EmployeeCount
    .
    .
    .
    NEXT N
RETURN

PrintChecks:
    .
    .
    .
RETURN
```

子程序CaleDeductions中的变量N和主程序中的变量N有着同样的名字，子例程并不把变量变当作局部变量，以至于事实上 N 将被子程序改变。结果是：在这个例子中，当程序流程最后返回到调用模块或主程序时，N 将不保持它的原来值 1。在

调用之后，经过 CaleDeducations 中的 FOR ... NEXT 循环 N 将包含值 EmployeeCount+1。这将导致错误的结果。

在过程和函数中把变量显式地说明为局变量的这种功能将减小上面那类错误发生的危险（可能）。因为过程和函数能把变量定义为局部变量而子程序不能．所以鼓励使用这种程序结构代替子程序或模块．同样，从这点出发，也希望把所有过程和函数中的变量都定义为局部变量，如果这是有益的话．

注意：在过程和函数中把变量定义为局部变量的功能是Turbo BASIC的特色,其他版本的 BASIC 未必有这种功能．

总而言之，局部变量是指那些包含在过程或函数之内不影响程序的其他模块中同名变量的变量．如果一个同名变量出现在主模块和一个或多个过程中，但是这些变量不是参数，那么他们将被视为完全独立的元素．在过程中对该变量的任何改变都不会影响此过程之外的同名变量，虽然在过程中这是缺省的，但最好还是把这些变量定义为 LOCAL．然而为了在函数中获得这种同样的独立性，必须把这些变量显式地定义为 LOCAL．

不要把局部变量表中定义的变量同参数混淆在一起，参数根据以前讨论过的规则被传送到子程序或从子程序中传回．局部变量是指那些仅在子程序内部使用的变量，它们同参数不一样，不能传递到调用模块或从调用模块传送出来．

§12.3.2 共享变量

共享变量是局部变量的反义词，我们曾经讲过，函数中的变量其缺省属性即为共享变量；但是，如果想把过程中变量变成共享变量，则必须进行显式说明．

想在过程或函数中使用该过程或函数之外的、赋予给某些变量的值时，就把这些变量定义为共享变量．同样，如果该过程或函数改变了一个共享变量的内容，那么这种改变永久的影响整个程序中该变量的值．这意味着在该程序中其他的部分这些变量的内容也将发生同样的变化．

例：请考虑下述过程中共享变量与局部变量的区别：

把变量定义为局部变量

```
'Main program goes here
T = 0
CALL Proc1(X)
PRINT "T is : "; T
     .
     .
     .
END
```

B10 8-32 ②

```
'Subprograms go here
SUB Proc1(X)
     :
     :
     T = 5
END SUB

'T may be defined as local

SUB Proc1(X)
LOCAL T

'If T is not defined as
'local in Example 1a
'LOCAL is the default anyway
```

打印结果如下:　T is 0

把变量定义为共享变量

```
'Main program goes here
T = 0
CALL Proc1(X)
PRINT "T is : "; T
     :
     :
END
'Subprograms go here
SUB Proc1(X)
    SHARED T
    :
    T = 5
END SUB

'If T is to be SHARED,
'SHARED must be explicitly
'stated as in the above
```

打印结果如下:　T is 5

数组传递

为将数组传递给过程，可按如下方法编程:

```
DIM  Array1(1:25)
CALL  Proc1
      ⋮
END

SUB  Proc
       SHARED  Array1( )
         ⋮
```

也就是说，为了传递整个数组，编写 SHARED Array( )；这个（ ）意味着将要传递一个数组。注意：数组可以传递给过程但不能传送给函数。

§ 12.3.3 静态变量

根据定义静态变量是局部变量，也就是说，在过程或函数之内它的内容不会影响程序中其他部分的任何同名变量，静态变量和局部变量之间的主要区别是：每调用一次子程序局部变量都被赋值为 0（字符串变量赋值为空 null）

，然而静态变量将保存它的内容，将它从该过程或函数的一个调用传送到另一调用。

例:

```
                        CALL  Proc1                              CALL  Proc2
                        ⋮                                        ⋮
                        CALL  Proc1                              CALL  Proc2
                        ⋮                                        ⋮
                    SUB Proc1                                SUB Proc2
x and y ──→         LOCAL x, y                               STATIC x, y
are set                   ⋮                                        ⋮
to 0 each                 ⋮                                        ⋮
time the                  x = x + 2         x = 1 the first ──→    x = x + 1
procedure                 ⋮                 time the procedure     ⋮
is executed               ⋮                 is called; x = 2       ⋮
                                            the second time the
                                            procedure is called,
                                            and so on
```

其中：每次执行过程Proc1时，x 和 y 被置成0。

第一次调用过程Proc2时，x = 1；第二次调用过程

Proc2 时，x = 2；以此类推。

如果想记录一个过程或函数被调用的次数静态变量显然是十分有用的。

如果没有对过程中的变量进行显式说明，那么它们不仅会被缺省地认为是局部变量而且还被缺省地认为是静态变量（STATIC）。另一方面，如果没有对函数中的变量进行显式说明，那么它们将被缺省地认为是共享变量。

编程指南

为了避免混淆和更好地编制程序，最好把过程和函数中的变量显式地说明为局部变量，静态变量或共享变量，这可使程序易于阅读，易于调试，易于维护。

而且，对于使用众多的变量的程序而言，在编程时用户不当心使用一个名字代表两个不同的变量的可能性总是存在的、通过把过程或函数中的变量指定为局部变量，静态变量或共享变量，可以减少这种错误的发生。

下面的图表简明扼要地列出了过程和函数之间的异同。

行　为	函　数	过　程
模块有返回值吗?	有	无（或：至少不必要）
怎样调用?	在表达式中调用自定义函数时，用"FN 标识符( )"；而调用预先定义的函数时，用"函数名( )"调用	CALL 语句
怎样传递参数	只能按值传递	既能按引用传递，又能按值传递
变量的缺省属性?	共享变量	静态变量
能否传递数组?	不能	能

自我测验(四)

阅读下面的程序，并回答问题 1—3

```
SUB Test
    LOCAL Number1
    PRINT Number1
    Number1 = Number1 + 1
END SUB
```

1. 第三次执行Test时，将打印什么?
2. 考虑下面的程序

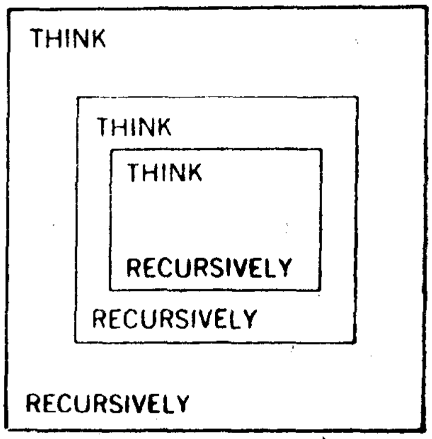
```
Number1 = 8
CALL Test
PRINT Number1
```

   CALL 之后将打印什么?
3. 假定Number1定义为STQTIC（静态变量），那么第三次执行Test时，将打印什么?
4. 过程中的变量其缺省属性是：（Local，Static，Shared）.
5. 函数中的变量其缺省属性是：（Local，Static，Shared）.
6. （T或F）过程、函数和子程序都是既能使用局部变量又能使用共享变量.
7. （T或F）可以把整个数组传递给过程，但不能传送组函数.
8. 静态变量和局部变量的区别是什么?
9. 输入一个数X，编写一个函数检查该数是奇数还是偶数. 如果该数是偶数给主模块返回代码 1；如果该数是奇数，给主模块返回代码 0. X 从主模块传送到函数，代码从函数传回到主模块，而且在函数中代码应定义为共享变量.
10. 编写一个程序完成与第9题同样的任务，但是在过程中要打印相应的"odd"或"even".

§12.4 递归

§12.4.1 递归的定义

递归是一种重要的功能强大的解决问题的工具，它在很多类型的程序中都经常使用. 递归过程或函数就是调用自身的过程或函数，如果该过程或函数改变了作为参数的变量的内容，那么它每次都可以使用不同的值被重复执行. 如图所示，用户可以这样想象递归过程或函数；在此，参数值逐渐减少，就象下图所示的一系列现象一样.



Turbo BASIC 允许过程和函数调它们自身．下面的两个例子是允许的:

例 1a: 一个调用自身的过程．

```
SUB A (parameter list)
    .
    .
    .
    IF ... THEN
       CALL A (parameter list)
    END IF
    .
    .
    .
END SUB
```

为了保证在逻辑上终止一个过程的重复执行，一般情况下，只有在遇到特定的条件时该过程才被重复执行．最典型的例子是，这个条件依赖于一个或更多的参数的值．参见下例:

例 1b: 另一个调用自身的过程

```
SUB A(X)
    .
    .
    .
    X = X - 1
    IF  X > 0 THEN
       CALL SUB A(X)
    END IF
END SUB
```

或:

```
SUB A(X)
    .
    .
    .

    IF  X > 0 THEN
        CALL SUB A(X - 1)
    END IF
END SUB
```

同样，在下面的程序中，第一个过程调用第二个过程，第二个过程也调用第一个过程。这在 Turbo BASIC 中也是允许的。

例 2: 过程 1 调用过程 2，过程 2 又调用过程 1:

```
SUB A (parameter list)
    .
    .
    .
    IF  ... THEN
        CALL B (parameter list)
    END IF
    .
    .
    .
END SUB
SUB B (parameter list)
    .
    .
    .
    CALL A (parameter list)
    .
    .
    .
END SUB
```

其中，过程A或B同样需要包含一些测试，以保证这些过程的重复执行在某一点被终止。

例 3 调用自身的函数:

```
DEF FNTest2(X)
    .
    .
    .
    IF ... THEN
        FNTest2 = ... FNTest2(X - 1)
    END IF
    .
    .
    .
END DEF
```

在此也不例外，FNTest2必须包含一个IF，满足条件时将终止该函数的重复执行。

例 4: 调用自身的子程序:

```
GOSUB Test3
.
.
.
END
Test3:
    .
    .
    .
    IF ... THEN
```

调用自身的子程序或子程序之间的互相调用称作"递归子程序(recursive subprogram)".

§12.4.2 Turbo BASIC 中递归使用举例

§12.4.2.1 计算阶乘

可以编写一个递归函数来计算 1–10 之间的数的阶乘. N 的阶乘定义为:

$$N! = +N \times (N-1) \times \cdots \times 1$$

该函数如下所示:

```
DEF FNFactorial#(N%)
    IF N% > 1 AND N% <= 10 THEN
        FNFactorial# = N% * FNFactorial#(N% - 1)
    ELSEIF N% = 0 OR N% = 1 THEN
        FNFactorial# = 1
    ELSE
        FNFactorial# = -1
    END IF
END DEF
```

为了计算一个输入的 1 到 10 之间的整数的阶乘, 编写如下的程序:

```
PRINT "Enter an integer from 1 to 10"
INPUT N%
PRINT FNFactorial#(N%)
```

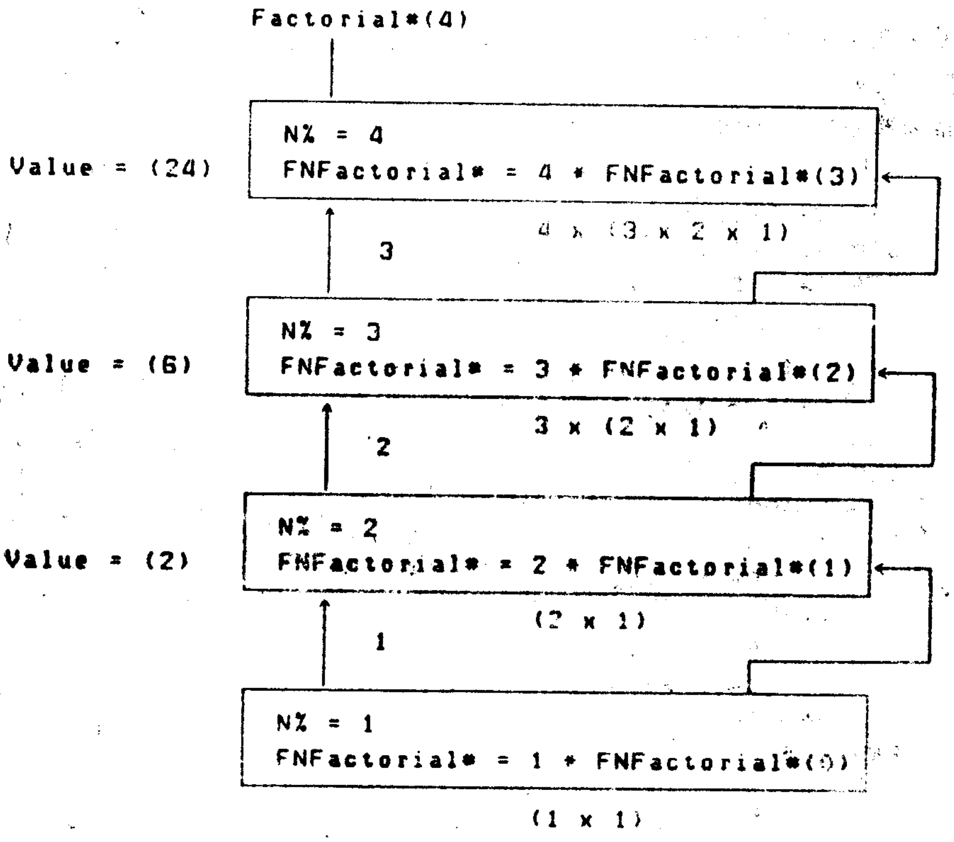
把上面这个例子和计算阶乘的非递归函数进行比较:

```
DEF FNFactorial2#(X%)  ''X% is an integer variable
    LOCAL I%, Total
    IF X% < 1 OR X% > 10 THEN
        FNFactorial2# = 0
        EXIT DEF
    END IF
    Total = 1
    FOR I% = 2 TO X%
        Total = Total * I%
    NEXT I%
    FNFactorial2# = Total
END DEF
```

注意：在大多数情况下，递归函数比非递归函数短。在该例中，FNFactorial#仅包含一个不带局部变量的 IF 块。就象在这个例子中所看到的那样，递归明显地降低了子程序的长度。

递归子程序的第二个特点是：它们多少有点复杂。

下面是一张递归如何工作的图示。假定 N%是 4。N% * FNFactorial (N%−1) 的乘法首先被执行，然后分别执行 N%＝3，N%＝2，N%＝1；执行时，从"底框"开始。



一个计算 N!的递归函数如下所示：

```
Factorial# = 1
INPUT N%
GOSUB Factor
PRINT Factorial#
END
Factor:
    IF  N% > 0 AND N% <= 10 THEN
        LET Factorial# = Factorial# * N%
        LET N% = N% - 1
        GOSUB Factor
    END IF
RETURN
```

§ 12.4.2.2 按反向顺序打印数值

考虑一个包含 25 个元素的数组:

```
OPTION  BASE  1
DIM  ARRAY1(25)
```

假定这些元素已经存在这个数组里，编写一个过程把这些元素例着打印出来，也就是说，按相反的顺序。

下面为递归的一个实例:

```
OPTION BASE 1
DIM Array1(25)
FOR I% = 1 TO 25
    INPUT Array1(I%)
NEXT I%
    CALL PrintOut(25)
END
SUB PrintOut(N)
    SHARED Array1( ) '( ) indicates an array is passed
    IF N = 1 THEN
        PRINT Array1(1)
    ELSE
        PRINT Array1(N)
        CALL PrintOut(N - 1)
    END IF
END SUB
```

按倒序打印 25 个数包含了对过程 PrintOut 的 25 次调用，每一次调用都使用不同的参数值，该过程每一次执行的返回顺序和过程调用的顺序正好相反。也就是说，首先从最后一次调用返回，然后再从倒数第二次调用返回. 依次类推. 这叫做 LIFO 通路–后进先出。

注意：总是能够用迭代的方法实现递归过程所执行的同样的操作集合。考虑下面例子:

```
FOR  N% = 25  TO  1  STEP  -1
      PRINT  Array1(N%)
NEXT  N%
```

§ 12.5 CHAIN（链接）语句

在这一章，重点讨论了自顶向下逐步求精的编程方法．其中，只有当主模块完成之后，才能编写子模块或从属模块．大而复杂的程序可以把子模块作为外部子程序从其他的程序中调进．外部子程序可以作为完全独立的程序读入．可以使用 CHAIN 语句把它们连接到主程序上，在外部子程序被装进执行的同时 CHAIN 语句将暂停主程序的执行．

CHAIN　文件说明符

其中文件说明符必须使用字符串格式．

下面是在主程序中使用 CHAIN 语句编程的例子．

例:

```
'Load and execute Module1 from the logged or default drive
CHAIN "Module1"

'Load and execute Module2 from the B drive
CHAIN "B:Module2"
```

CHAIN 语句中定义的每一个模块都被使用选择命令中的"编译成链接文件"建立，这是当建立链接文件时编译器使用的扩展功能．

CHAIN 语句中使用的文件名必须是 1-8 个字符，如果没有规定扩展名，就假定扩展名是.TBC.因此，在上面的例子中，Module1.TBC 和 Modeue2.TBC 是被作为独立的程序进行链接的外部子程序的全名．如果外部子程序的文件说明符有不同的扩展名，那么它必须在 CHAIN 语句中显式地说明．因此，如果想链接驱动器 A 上的 Module3.EXE，则编程如下:

```
CHAIN "A:Module3.EXE"
```

只有当 Turbo BASIC 主程序直接从 DOS 执行时，链接才能够被执行．也就是说，全部调试完成后，Turbo BASIC 主程序必须编译成带.EXE 扩展名的文件然后执行．为了把主程序编译成.EXE 文件．选择 Options 命令中的"Compile to EXE file"选项．

为了把变量从主程序或链接程序传送到叫做被链接程序的子程序．COMMON 语句总是连同 CHAIN 语句一起使用．

考虑下例:

```
COMMAND EmpName$, Hours, Rate
CHAIN "B:Module4"
```

COMMON 语句把上面定义三个变量的内容传送到 Module4，然后执行程序 Module4。

如果该 COMMON 语句被省略，那么链接程序或主程序中不会有变量传递到被链接程序。COMMON 语句的格式如下：

COMMAND　变量[, 变量]

从链接程序传送到被链接程序的每一个变量可以是字符串类型，数值类型或数组。

一个匹配的 COMMON 语句必须出现在被链接的程序中，在该 COMMON 语句中给予变量的名字无关紧要。但是变量的个数和类型必须和链接程序中变量的个数和类型一一匹配。

例：　程序链接

```
Main Program
   .
   .
   .
   COMMON EmpName$, Hours, Rate
   CHAIN "B:Module3"
   .
   .
   .
Module 3
   COMMON E1$, H1, R1
   .
   .
   .
   END
```

执行 Module3 后，程序流程返回到 DOS 状态。

§12.6　本章辅导

§12.6.1　本章小结

一、函数的定义

A. 函数返回给调用模块一个值（参数）

B. 函数可用来作为自顶向下的模块结构。

C. 预先定义的函数：

1. 预先定义的函数已经成为Turbo BASIC语言的组成部分。

2. 预先定义的函数通常用在赋值语句中作为表达式的一部分，或者用在

PRINT 语句中。

3. 请参见P635．P538中预先定义的数值型和字符串型函数表。

D. 专用函数可由用户自定义。

1. 自定义函数用FN标识符调用

2. 函数以DEF FN函数名开始，以END DEF 结束。

E. 函数的特征:

1. 函数一般放在END语句之后。

2. 函数可以编成一行，也可以编成多行。

a. 多行函数需要END DEF 范围终止符。

b. 单行函数不需要范围终止符。

c. 可选择的参数最多可有16个参数，当调用函数时，它可以把值传递给函数。

d. EXIT DEF 可用来提前终止函数的执行。

e. FN标识符＝表达式定义了要传送给调用模块的值。

3. 函数总是使用按值传递方法传递参数。

4. 函数既能返回数值型的值，又能返回字符串类型的值。

二、在过程和函数中说明变量

A. 在Turbo BASIC中，用在任何可执行的语句中的变量都被隐含地说明:

1. 在过程或函数定义的局部变量独立于调用模块，也就是说，在过程或函数内赋给该变量的值不会传递给调用模块。

2. 在过程或函数中定义的共享变量可以改变主模块或调用模块中该变量的内容，当然它也可被改变。

B. 过程或函数中的局部变量不影响程序中其他模块内的同名变量。

1. 在过程内说明的变量其缺省属性是局部变量。

2. 在函数内的变量其缺省属性是共享变量。

3. 既可以把变量显式定义为共享变量，也可以把它显式说明为局部变量。

4. 变量定义（说明）[SHARED 或LOCAL]应紧跟在SUB 或DEF 语句之后。

5. 子程序中说明的变量都是共享变量，不能说明为局部变量。

6. 多次执行子程序时局部变量赋值为空或0。

C. 在子程序中对共享变量所做的变化将永久地改变整个程序中这些变量的内容。

D. 静态变量是一种局部变量，它把它的值从子程序的一个执行传递到该子程序的下一次执行．也就是说，静态变量不同于局部变量，每次执行该子程序对变量都赋值为 0 或空。

E. 把函数或过程中的变量显式地说明为局部变量，共享变量或静态变量总是最安全的。

三、递归

A. 递归过程或函数是指调用自身的过程或函数.
  1. 递归使得过程或函数用不同的值被重复执行.
  2. 一般而言，过程或函数只有当特定的条件满足时才重复执行（这是为了防止无限循环).

B. 子程序也可以是递归的.

C. 一般而言，递归会减少子程序的长度，但是它经常是更难编写.

四、CHAIN 语句

A. 在装入外部子程序的同时CHAIN语句使得主程序暂停执行.

B. 只有当程序被编译成.EXE文件执行时才能执行链接过程.

C. COMMON语句用来把变量从主程序传送到子程序.

§12.6.2 本章自测

请写出下列程序执行时的结果:

1

```
'Illustrating the RIGHT$ Function
Animal$ = "Hippopotamus"
L = LEN(Animal$)
FOR J = 1 TO L
    Part$ = Right$(Animal$, J)
    PRINT Part$
NEXT J
END
```

2.

```
'Illustrating the LEFT$ Function
Animal$ = "Hippopotamus"
L = LEN(Animal$)
FOR J = 1 TO L
    Part$ = LEFT$(Animal$, J)
    PRINT Part$
NEXT J
END
```

3. 编写一个Turbo BASIC程序，把在一行上输入的一段英文正文翻译成Prg latin, 在 pig latin 中每个词的第一个辅音字母被移到词尾并且要加上后缀 ay，例如，在 pay latin 中 Turbo 应是 urbotay.

4. 编写一个Turbo BASIC 程序用来和用户玩游戏. 让计算机选择一个1-100 之间的随机数，然后给用户 10 次机会猜这个数, 对于不正确的猜测让计算机回答" Toohigh "或"Too low".

解释下列程序执行的结果（问题 5–7）:

5.
```
LET Q$ = "        "
READ P$, A
DO  WHILE P$ <> "END"
    LET A = INT(A)
    PRINT LEFT$(P$ + Q$, 8);
    PRINT RIGHT$(Q$ + STR$(A), 5)
    READ P$, A
LOOP
DATA "BOB", 226.42
DATA "RICHARD", 15.6
DATA "END", 0
END
```

6.
```
C$ = "
READ X
DO  WHILE X <> 0
    LET I$ = STR$(INT(X))
    LET E$ = STR$(INT(X * 100))
    LET F$ = RIGHT$ ("00" + E$, 2)
    PRINT RIGHT$(C$ + I$ + "." + F$, 10)
    READ X
LOOP
DATA 22.63, 182.831, 7.4
DATA -14.63, 0
END
```

7.
```
PRINT "123456789012345"
READ X
DO  WHILE X <> 0
    LET J$ = STR$(INT(X))
    LET E$ = STR$(X - INT(X))
    LET F$ = MID$(E$, 3, 2)
    PRINT TAB(8 - LEN(J$)); J$ + "." + F$
    READ X
LOOP
DATA 32.463, 72.862
DATA 6.3, -26.72, 675, 0
END
```

8. **指出下面每个PRINT语句的显示结果:**

a. PRINT ABS(13.8)

b. PRINT ABS(−6.3)

c. PRINT INT(39.3)

d. PRINT SQR(16)

e. PRINT MID $ ("Somebody", 3, 2)

指出下面各例的结果（9～13题）

9. PRINT INSTR(STR$(4, 12), ", ")

10. PRINT "**" + STR$(2, "*****") + LEFT$("*****", 2)

11.
```
C$ = "COMPUTERS"
Length = LEN(C$)
N = 1
DO  WHILE N <= Length - 4
    PRINT N
    N = N + 1
LOOP
```

12.
```
B = 12.345
B$ = STR$(B)
P$ = MID$(B$, 2, 4)
L = LEN(P$)
PRINT L
PRINT P$
```

13.
```
P = 1
DO  WHILE P <= 3
    PRINT MID$("ABCDEFG", 3, P)
    P = P + 1
LOOP
```

14. 编写一个程序用来玩游戏"carps"。这个游戏用两个骰子来玩，每个骰子包含1到6个点，游戏一开始，玩的人滚动两个骰子．如果两个骰子的总数为7或11，那么就取胜.如果总数为2.3或12，他或她就输了．如果总数是4.56.8.9或10，玩的人不赢不输．而且，这个总数叫做玩者的"point(点)"，程序（游戏）继续，玩者继续掷骰子直到原来的点数（总数）掷出，在此点数他或她取胜，或者掷出7来，这意味着玩者输了．一旦赢了或输了，新的游戏也就开始了．

    用户编写的程序应该询问游戏玩的次数，并输入它．而且应该输出赢输的次数．

15. 编写一个程序从INPUT语句中接受一个字符串并把所有的大写字母变成小写字母，小写字母和其他字符不变．用户编写的程序应打印出输入字符串的原始形式和变化后的正文．

    注意：完成这个任务有很多方法，但不能使用函数 CASE 或 LCASE $．

§12.6.3 实例

计算弗班那塞(Fibonacci)数

下面是一个计算机科学家通常用递归函数解决的一个常见问题：

弗班那塞数是一个增长非常迅速的整数序列，有人说它们起初用来描述兔子繁殖速度；当然，这很可能是一个神话．弗班那塞序列如下所示：

1, 1, 2, 3, 5, 8, 13, 21, 34,…

其中: 第三个数等于前两上数的和，第四个数等于第二和第三个数的和，依次类推. 可用数学关系描述如下:

$$
\begin{aligned}
N_1 &= 1 \\
N_2 &= 1 \\
N_3 &= N_1 + N_2 \\
N_4 &= N_2 + N_3 \\
&\vdots \\
N_n &= N_{n-2} + N_{n-1} \qquad \text{其中: } n > 2
\end{aligned}
$$

可以使用 DO WHILE ... LOOP 逻辑控制结构找到第 M 个弗班那塞数. 在此，M 是键入的数值:

```
CLS
LET Count = 0
LET N1 = 1
LET N2 = 1
PRINT "Enter the # of Fibonacci numbers to calculate after 1,1";
INPUT M
PRINT N1
PRINT N2
DO WHILE Count   M
        NewNumber = N1 + N2
        PRINT NewNumber
        N1 = N2
        N2 = NewNumber
        Count = Count + 1
LOOP
PRINT "End of Job"
END
```

Or, we could use a recursive procedure:

```
DEF FNRecursiveFib#(N%)
    IF  N% < 1 THEN
        FNRecursiveFib# = 1
    ELSEIF N% = 1 OR N% = 2 THEN
        FNRecursiveFib# = 1
    ELSE
        FNRecursiveFib# = FNRecursiveFib#(N% - 1)_
             + FNRecursiveFib#(N% - 2)
    END IF
END DEF
```

§ 12.6.4 复习题

I 综合题

1. 过程和函数之间的主要区别是什么?
2. 函数的两种类型是________函数和________函数.

3. 编写一个函数计算任何数的四次幂.
4. 程序定义多行函数以________开始，以________结尾.
5. 用________可以在一个计算执行完之前提前退出函数.
6. （T或F）函数将改变在调用模块中定义的参数的内容.
7. （T或F）函数既可以返回数值类型的值，也可以返回字符串类型的值.
8. （T或F）如果一个函数改变了同名变量的值，那么主程序中的变量随将改变.
9. 计算1-100（包含100在内）之间的随机数.
10. 编写一递归函数持续对某数加倍到结果大于5000为止.
11. 指出当执行下列PRINT语句时显示的结果:

```
a. PRINT  INT(-62.14)
b. PRINT  ABS(-16.3)
c. PRINT  LEFT$("Anybody", 2)
d. PRINT  RIGHT$("Everybody", 4)
e. PRINT  INSTR("Some one has it", "on");
f. PRINT  LEN("Good News!")
```

12. 编写一程序输入一串字符然后反序打印该字符串（如，输入ABC应打印CBA）
13. 编写一个程序输入一段正文（在75个字符之内），统计该正文中"the"出现的次数（注意：使用函数 INSTR 找到词"the"并确信它不是另一个词的一部分，也就是说，在"the"后没有紧接跟着字母）.
14. 编写一个程序统计从键盘输入的一段正文中单词和句子的个数. 假定每个空格标志着一个词的结束，句号标志着一个句子的结束. 假字，输入正文的末尾用斜杠（/）标识.
15. 编写一个交互程序接收一个1到100之间的数，让计算机猜输入的数，不能超过7次，用户给计算机唯一的线索就是响应指出猜的值太大或太小.
16. 编写一个程序读入一行正文并输出出现两次的字母. 也就是说，在一行之内一个字母出现了两次.
17. 修改问题16以便该程序能输入出包含双字母的具体的词，使用递归方法.
18. 编写一程序输入一行正文并打印所有整数值和整数值出现的个数.
19. 编写一个程序输入一行正文并打印每个单词的平均包含的字符个数.
20. 编写一个程序，打印一行正文后跟元音的全部辅音字母.

II 验证数据有效性

修改实例，使之包括适当的数据确认代码:

(1) 测试所有的错误;

(2) 打印总的控制表（处理记录时，所遇到的错误总批数）.

§ 12.6.5 调试练习

改正下列程序中的错误:

1. ```
   K$ ="Turbo Basic is fun.")
   J$ =INSTR(K$, "basic")
   ```
2. ```
   A$ =CINT (−435.746)
   ```
3. ```
   S$ ="Fourscore and seven years ago"
   L$ =MID$ (S$, 15, 5)
   ```
4. ```
   SUB  AddIntegers
           LET  Total = Total + N
           GOSUB  AddIntegers
   END  SUB
   ```
5. ```
   IF  NOT  EOF(#1)  THEN
           ⋮
           ⋮
   END  IF
   ```
6. ```
   P$ =INSTR (1, Text$, "A"
   ```

§ 12.6.6 编程题

1. 从键盘接收三个单词，并选出其中最长者.
2. 用随机数模仿骰子，统计50对骰子出现7及11的次数（注意：用自定义函数来产生1～6 的随机数）
3. 编程计算本金翻一番所需的月数，其年利率分别为：7%、8%、9%及10%。计算时间的公式如下：

   N=log2／[M(log (1 + R／M)]

   其中：N 为年数，R 为利率，M 为转换周期数.
   要求将结果精确到两位小数.
4. 从键盘输入字母并赋给变量，使用MID $ 函数及连接操作符+，使程序能挑选和组合出用户的名字，并打印出来.
   例如，要打印的名字为 TOM，则可编成：

   ```
   MID$ (ALPHA$, 20,1)  + MID$ (ACPHA$, 15,1)  + _
   MID$ (ALDHA$, 13, 1)
   ```
5. 编程计算周工作时间为39的工人工资，超时工资加倍，扣除28%的个人所得税，1%的会费，伙食费 8%。请使用三个函数:

a. 一个函数用于计算包含超时工资的工资。

b. 一个函数用于计算不包含超时工资的工资。

c. 第三个函数计算扣除上述费用以后的工资。

第十三章　文件处理和数据库管理

本章要求:

1. 掌握文件处理的概念.
2. 了解顺序文件与随机文件的区别.
3. 掌握创建、修改顺序文件和从中生成报表的方法.
4. 掌握创建、修改随机文件和从中生成报表的方法.

§13.1　主文件处理的系统概述

§13.1.1　什么叫主文件

在第六章，已经介绍了文件处理的概念. 文件(file)是一个关于专用应用程序的数据集. 每个文件是由记录组成; 而这些记录又可分成信息字段, 如 EmpName $ 和 Salary. 现在讨论在磁盘上怎样建立顺序文件, 怎样应用这些文件来打印详细报表, 总结报表和异常报表.

在这节里, 将更详细地考虑文件尤其将重点集中在主文件(Master file)处理上, 主文件是一个关于专用应用程序的主要数据集. 一个公司有关于工资名单、应收取账目、应付出账目、销售量和库存清单的主文件. 工资名单主文件可用来开支付支票. 应收取账目主文件可用来产生一张账单, 等等.

主文件存贮在硬盘中, 或者, 对一小的应用程序, 可存在磁盘上.

一般有两种类型的主文件:

(1) 顺序文件;
(2) 随机存取文件;

顺序文件是指通常按顺序处理的文件. 即, 读入且处理完记录 1, 接着读入和处理记录 2, 依次进行下去. 为了存取第 986 号记录, 必须读过前面 1～985 号记录. 随机存取文件又称随机文件, 其记录可直接存取, 而不需要从文件开头起读过前面的记录. 当不需要按顺序处理记录时, 通常建立随机文件. 例如:有一个应收取帐目主文件, 被用来回答关于收支平衡的查询, 那么此文件必需是随机存取的. 因为不能预测哪一用户将提出查询, 所以需要能够从这个文件中随机存取记录.

§ 13.1.2 典型的主文件过程

不管主文件是顺序存取还是随机存取，最一般的主文件过程包括：

§ 13.1.2.1 建立一个主文件

当完成一个新的计算机化系统或第一次应用该系统时，对每一应用程序区域都必须初始建立一个主文件，这一"建立"程序，利用键盘终端或者其它数据输入装置来接受所有主文件数据．然后，将这些数据记录在磁盘上，形成一个新主文件．主文件的初始建立是一遍过程，也就是说，一旦建立了主文件，就需要通过其它的过程或者程序来更变它．

建立主文件程序的主要目标首先是确保数据完整性．也就是说：只有当主文件中包含的是有效，可信数据，它才有用．因此，必须设计建立过程，以便使输入错误减到最少，而后建立程序还必须检查输入变量以确保：

(1) 变量处在正确的格式；

(2) 变量处在预定定义的范围内(例如，如果0001－9825代表了变量Actions的范围的话，就必须检查变量 Actions 的值是否在此范围内)；

(3) 变量所包含是有效数值(例如，对变量Sex，必须检查以确信其包含的不是"M"就是 F")．

主文件建立时，同时产生控制列表(Control listing)和记录跟踪(audit trail)，用其来打印新主文件的数据任何控制总数都是必要的．通过利用其它附加技术来检查和检验控制列表和记录跟踪使其错读减到最少．

§ 13.1.2.2 建立更变记录的细目文件

在建立主文件以后，必须编写一个程序用来存贮针对主文件的变化，将更变记录存贮在一单独文件中，此文件叫细目文件(Transaction file)．例如,应收取账目主文件的更变可能由销售记录和信贷记录组成；工资名单主文件的更变就可能由姓名记录和工资记录组成；等等．这些更变记录都将存贮在磁盘上的细目文件中，正如建立主文件一样，细目文件也应首先必须确保数据的完整性．有效细目数据方能使出错的机会降到最小．

§ 13.1.2.3 修改主文件

修改顺序和产生一个新主文件，都是通过读当前主文件和细目文件来周期性地修改顺序主文件来完成的．因为在修改过程处理这二个文件中，将所有记录和起作为一批，所以叫做批处理(Batch Processing)．由于顺序主文件的批处理周期性地执行修改，所以其效率很高．但是，此处理并不及时，因为实际上对主文件的更变是在一个固定间隔时间内进行的．

在要更变记录时，随机存取文件的修改是交互性进行的．也就是说：当用户购买时，操作员通过键盘或者终端可随机地存取这一用户的记录，输进购买数量，立即修改这一记录．这类修改是非常及时的．但是，因为要求在任何时候用户申请购买，都能够对主文件和修改程序进行处理，所以，费用是较昂贵的．

当文件不要求随机存取和一直保持最新时，我们可采用顺序主文件，利用批处理方式进行修改．我们将在以下的三节(§13.3—§13.5)中将讨论顺序修改，而对用交互处理来修改时，必须建立随机存取文件．将在第5节中讨论这一过程．

§13.1.2.4 来自主文件的报表

维护一个主文件的基本目的就是存贮数据，以便能够向用户提供有意义的输出内容，这就叫做报表．不论何时需要，报表或者输出应列成表格，都能够按照一定基本规则或要求准备好．销售报表、账号和工资表都是从主文件中输出的例子．这些都应按照基本规则列表准备；而对某些特殊要求，如显示出今年达到65岁的雇员名单，则需按特定的要求准备．

报表在输出要求方面易发生变化．实际上，用户关于输出报表的要求有时是很难预测到的．

在第六章和第七章中，讨论了利用细目打印、异常打印和按组打印技术准备来自顺序文件的列表报表．以后，我们将讨论来自于随机存取文件的报表．

§13.1.3 文件处理程序和软件包

数据库管理系统(DBMS)中有用来对各种类型文件进行建立，修改和报表的程序．可是，当某公司有特殊的要求时，诸如用 Turbo Basic 语言写出的程序将常被用来代替数据库管理系统对文件进行建立，修改和报表．

§13.2 修改顺序文件

§13.2.1 顺序文件结构概述

最简单的磁盘文件结构就是顺序的．存贮在顺序文件中的记录就是按某一关键字段如：用户编号、零件编号和雇员编号等顺序输入的．因此，相对比较容易确定某一给定的记录．例如，假设雇员编号是连续的，那么编号名 00883 的记录其物理定位就在编号名 00882 和 00884 之间．

顺序文件必须按照顺序处理，这就是其主要不足之处．如果用户要访问一条雇员编号为 23000 的记录，计算机就必须读过前 22999 条雇员记录．以后将看到，若采用随机文件，记录可以直接访问而不需要按顺序去读所有的记录．

顺序文件必须按顺序处理，所以当所有的记录或者大部分记录需按顺序访问时，最好采用顺序文件进行批处理操作．

§ 13.2.2 顺序主文件和细目文件

正如在第六章所见的那样，顺序文件中的记录是由用逗号分隔开的字段串来建立的。顺序文件中各记录间是通过加换行符来分开的。

例: 应收取账目记录

账号	顾客 姓	顾客 名	结余金额
0001,	"Newman"	"Paul"	100.75
0002,	"Evans"	"Linda"	55.25

图 13.1 说明了一段可用来建立应收取账主文件的程序的例子.

EDIT 窗口

```
'This program creates an accounts receivable master file
CLS
OPEN "ARMaster.Dat" FOR OUTPUT AS #1
StillMoreRecords$ = "Y"
DO WHILE StillMoreRecords$ = "Y"
        PRINT "Enter Acct No:"
        INPUT AcctNo
        PRINT "Enter Customer's Last Name:"
        INPUT CustLastName$
        PRINT "Enter Customer's First Name:"
        INPUT CustFirstName$
        PRINT "Enter Balance Due:"
        INPUT BalDue
                'Validate routines would go here
        WRITE #1, AcctNo, CustLastName$,CustFirstName$,BalDue
        PRINT "More Input? (Y/N)"
        INPUT StillMoreRecords$
LOOP
CLOSE #1
END
```

RUN 窗口

```
Enter Acct No:
? 0001
Enter Customer's Last Name:
? NEWMAN
Enter Customer's First Name:
```

```
? PAUL
Enter Balance Due:
? 100.50
More Input? (Y/N)
? Y
Enter Acct No:
? 0002
Enter Customer's Last Name:
? EVANS
Enter Customer's First Name:
? LINDA
Enter Balance Due:
? 55.00
More Input? (Y/N)
? Y
Enter Acct No:
? 0003
Enter Customer's Last Name:
? MOUSE
Enter Customer's First Name:
? MICKEY
Enter Balance Due:
? 44.00
More Input? (Y/N)
? Y
Enter Acct No
? 0004
Enter Customer's Last Name:
? WASHINGTON
Enter Customer's First Name:
? GEORGE
Enter Balance Due:
? 44.25
More Input? (Y/N)
? Y
Enter Acct No:
? 0005
Enter Customer's Last Name:
? MURPHY
Enter Customer's First Name:
? EDDIE
Enter Balance Due:
? 35.00
More Input? (Y/N)
? N
```

输出文件：ARMaster.Dat

```
1,"NEWMAN","PAUL",100.5
2,"EVANS","LINDA",55
3,"MOUSE","MICKEY",44
```

```
4,"WASHINGTON","GEOGER",44.25
5,"MURPHY","DEEIE",35
```

图 13.1　建立应收取账目主文件的 Turbo Basic 程序样例

若要建立一个主文件，则需一些主要的输入／输出指令，如: OPEN-... AS #1, WRIT#1，和 CIOSE#1 对于打开，写入或关闭同一文件，其文件号如在该说明中的 #1 必须是相同的。

我们会看到 WRITE#(文件号)语句可将信息输出到顺序文件中; 以后，用 INPNT 语句可从文件中读取信息，但其必须包含有与 WRITE 语句中完全相同的字段和格式.

向顺序文件中输出数据语句

```
WRITE #1, MAcctNo, MLastName$, MFirstName$, BalDue
```

在不同的程序中，从同一个顺序文件中输入数据的语句

```
INPUT #1, MAcctNo, MLastName$, MFirstName$, BalDue
```

所以，若要建立一个顺序文件，则应:

1. 打开文件用作输出。
2. 用WRITE#(文件号)语句在文件中建立记录.
   a. 用回车换行符作为写在磁盘上的每条记录的结果标志.
   b. 每条记录是由不同长度的字段组成，字段与字段之间用逗号分隔开.
   c. 每个字段的分界符双引号是由计算机自动加上的，在将建立主文件的数据键入时，不必输双行号.
3. 以后，我们从同一文件中读取数据时，INPUT#(文件号)语句中的变量和顺序都必须和 WRITE#(文件号)语句中的相同.
4. 在作业结果后，对每一文件都必须编上CLOSE语句.

用 Turbo Lasic 建立的顺序文件，由于使用了通用的 ASCII 字符标准，因而能够被别的程序甚至于数据库管理系统软件所识别.

用 OPEN 打开的文件名字，必须由 1～8 个字符组成; 这些字符可以是字母、数字和一些特殊字符(如 $，#，@，—．{ }），但第一个字符必须是字母．在三个字符组成的扩展名前允许有一个句号，用作输出的数据文件其扩展名常取・DAT.

§ 13.2.3　修改过程

回顾一下，修改过程就是使主文件是最新的．修改磁盘顺序名文件，要用到以下的三

个文件: 更变记录的细目文件，原主文件或旧主文件和新主文件．虽然不能够直接修改原主文件或旧主文件，但是常常宁愿保留它用做备份．这样的话,如果新主文件出错或者受列破坏，还可以用原(旧)主文件重新建立新主文件．在实际修改主文件时，建议还有第四个文件:打印文件．可用它来建立控制列表和记录跟踪．控制列表和记录跟踪将列出:

(1) 主文件的更新内容;
(2) 处理过程中遇到的错误;
(3) 用于控制和检查目的的总数;

由于用户已经熟悉了数据打印方法，所以在此省略了对控制列表的讨论，以便将着重点放在顺序修改过程中．可是，请记住: 所有"现实"主文件的修改都将产生一个用于控制和错误检查目的的列表．

§ 13.2.3.1 在批处理操作中采用输入主文件、输入细目文件和输出主文件

§ 13.2.3.1.1 输入主文件

在前一修改周期中，输入主文件是最新的．当其在第一次建立时，叫作 NewMstr 文件; 对下一个更新周期来说，文件 NewMstr 就成了 OldMstr 文件．在修改已后，建立一个独立的 NewMstr 文件，它将包含有文件 OldMstr 的数据和所有所有的更变．假设每周进行一次修改过程，那么，在修改之前，必须将上周的文件名字 NewMstr 更变成文件名字 OldMstr，被更名的 OldMstr 文件将不包含有发生在此周中的各项细目．

§ 13.2.3.1.2 输入细目文件

细目文件中包含有用于修改 OldMstr 文件的数据．当文件 OldMstr 建立时，输入细目文件包含有从上一修改周期起的所有更变，称这种文件为 TransIn 文件，并用其来修改 OldMstr 文件．

§ 13.2.3.1.3 输出主文件

输出主文件是指以修改过程为结果而形成的一个新主文件．输出主文件将包括有来自文件 OldMstr 和文件 TransIn 中的数据．正如以前所提到的，我们称其为 NewMstr 文件．当开始进行下一周的修改时，NewMstr 文件将更名为 OldMstr 文件．如前所述，在修改开始之前必须完成文件名的更变．上述所有的文件不是装在硬盘中就是装在软盘中．

考虑每周的修改:

第 1 周: 建立应收取账目主文件

```
OPEN "NewMstr.Dat" FOR  OUTPUT  AS #3
  ⋮
WRITE #3, MCustNo, MLastName$, MFirstName$, BalDue
```

注意: 数据文件需加扩展名 .DAT

第 2 周: 在第 2 周, 应收取账目主文件 NewMstr 将变成 OldMstr。

在第 2 周中，可用 DOS 将文件名由 NewMstr 更换成 OldMstr。假设上周的 NewMstr 文件在 A 驱动器中，我们可按下述步骤更换文件名.

```
A> REN  NewMstr.Dat   OldMstr.Dat
```

在将 NewMstr.DAT 文件变成 OldMstr 之前，磁盘上的任何先前的主文件或被删除或被更名，在 A > 提示符下键入:

```
A> ERASE  OldMstr.DAT
```

即可删除先前的 OldMstr 文件.

另外，在 Turbo Basic 程序本身中，可用:

```
KILL  "OldMstr.DAT"
```

删除先前的 OldMstr.DAT 文件. 在程序中，可以通过以下编码来更换文件名:

```
KILL  "OldMstr.DAT"
NAME  "NewMstr.Dat" AS "OldMstr.Dat"
  ⋮
```

如果 NewMstr 文件或者 OldMstr 文件不在当前的磁盘驱动器中，就必须预先打上驱动器代号和冒号. 在 OPEN 语句中，驱动器代号必须包括在引号中(如: "C:NewMstr.Dat"). 为了修改 OldMstr 文件，在将前一修改周期中的 Newmstr 文件更名成 OldMstr 文件之后，必须包括如下过程:

```
OPEN "OldMstr.Dat" FOR  INPUT  AS #1
OPEN "NewMstr.Dat" FOR  OUTPUT  AS #3
  ⋮
INPUT #1, MAcctNo, MLastName$, MFirstName$, BalDue
  ⋮
WRITE #3, MAcctNo, MLastName$, MFirstName$, BalDue
```

从文件号#1(OldMstr)中输入时，用到的所有字段名在向 Newmstr 文件中写时，都

可能用到．若使用同样的字段名字，会保证字段在文件 NewMstr 中和文件 OldMstr 中的位置完全相同．用 M 做为字段名的前缀，指明它们是在主文件中:

```
INPUT #1, MAcctNo, MLastName$, MFirstName$, BalDue
    ⋮
WRITE #3, MAcctNo, MLastName$, MFirstName$, BalDue
```

但是，由于这两个文件中指定的字段，在它们各自的程序中都指定有一定的含意，所以，我们可用下面的字段名来替代:

```
OPEN "OldMstr.Dat" FOR INPUT AS #1
OPEN "NewMstr.Dat" FOR OUTPUT AS #3
INPUT #1, AcctNoIn, LastNameIn$, FirstNameIn$, BalDueIn
AcctNoOut = AcctNoIn
LastNameOut$ = LastNameIn$
FirstNameOut$ = FirstNameIn$
BalDueOut = BalDueIn
    ⋮
' Steps are required to update BalDueOut before writing NewMst
WRITE #3, AcctNoOut, LastNameOut$, FirstNameOut$, BalDueOut
```

在以后的说明中，可以用赋值语句将 AcctNoIn、LastNameIn$ 等拷贝成 AcctNoout 和 LastNameOut$．

建议在新、旧主文件中都用同一字段名，这样可避免使用赋值语句；但是，如果用户宁愿使用后缀如 In 或 Out 来区别字段名，并使用赋值语句将输入字段的内容拷贝到输出字段中去，这也中可取的．后一技术可能使程序更加清析明显，但这需要增加编码和执行时间来将输入区域移到输出区域中去．

系统框图图 13.2 总结了用于修改过程中的文件．

§ 13.2.3.2 顺序修改记录次序

如前所述，在修改周期中，OldMstr 文件所包含的主要信息都是完整最新的．在文件 TransIn 中则包含有前修改期起所发生的所有细目或者更变，这些细目或更变必须和主文件相结合且保持为最新的．文件 NewMstr 中除了包含所在 TransIn 文件中的更变外，还包括所有 OldMstr 文件中的数据．对下一修改周期来说，NewMstr 文件将变成 OldMstr 文件，所以文件 NewMstr 和文件 OldMstr 具有相同的格式是必要的．

在顺序主文件中，所有记录都是根据关键字段(Key field) 如: 账号、社会保险号、零件号等来按顺序排列的．不同的主文件有不同的关键字段．关键字段是区分各个主记录的唯一标识．在顺序文件中，若要修改记录，那么在细目文件中包含的更变记录也必须按照同一关键字段顺序排列．因此，细目记录也必须用关键字段，比如账号来区别．

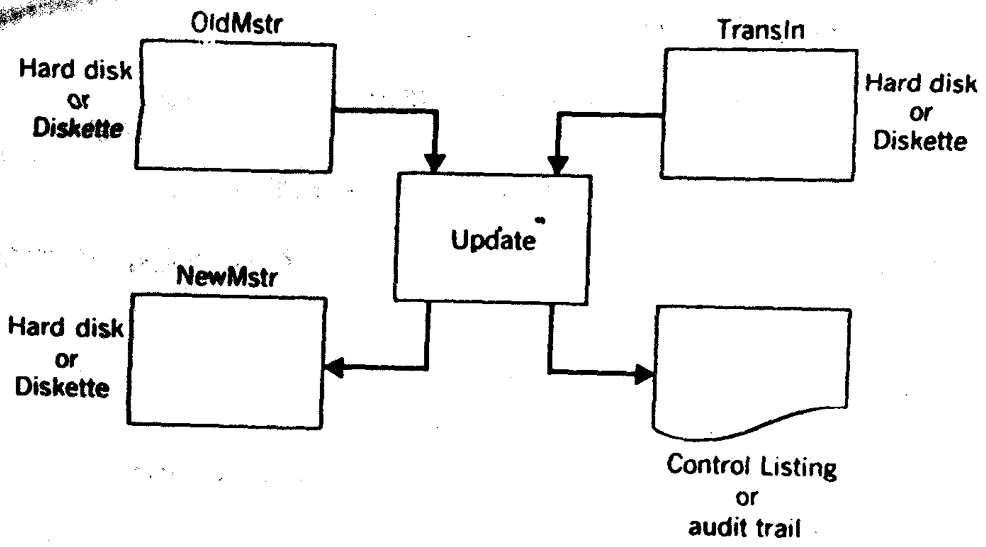


图 13.2　用于修改过程中的文件系统框图

§ 13.2.3.3　修改程序中所需的处理

下面来考虑顺序应收取账目主文件的修改，在 OldMstr 文件中用来识别记录的关键字段是账号，在程序中用 MactNo 来表示主记录账号．在应收取账目主文件中所有记录是按 MaxctNo 来排序的．图 13.1 中将应收取账目主文件更名建立一个 ARmaster.Dat 文件，以便将 OldMstr 文件用作修改用．

细目文件将包含有自前修改过程发生起的所有细目，这些细目都被传送到主文件中，细目文件也是以账号作为关键字段，叫作 TAcctNo，来表示细目账号，TransIn 文件中的记录就是按 TAcctNo 来排序的．

这两个输入文件格式如下:

OldMstr 记录

(按 MAcctNo 的顺序排列)

MAcctNo, MLastName$, MFirstName$, BalDue

NewMstr 记录

(按 TAcctNo 的顺序排列)

TAcctNo, TLastName$, TFirstName$, AmtOfSale

修改过程完成以后，文件 NewMstr 将成为当前的主应收取账目文件，其字段的顺序必须要与 OldMstr 文件中相同。将 OldMstr 文件中的字段名用在 NewMstr 文件记录中，这样做，会更加精确地将数据从一个文件复制到另一文件中去。

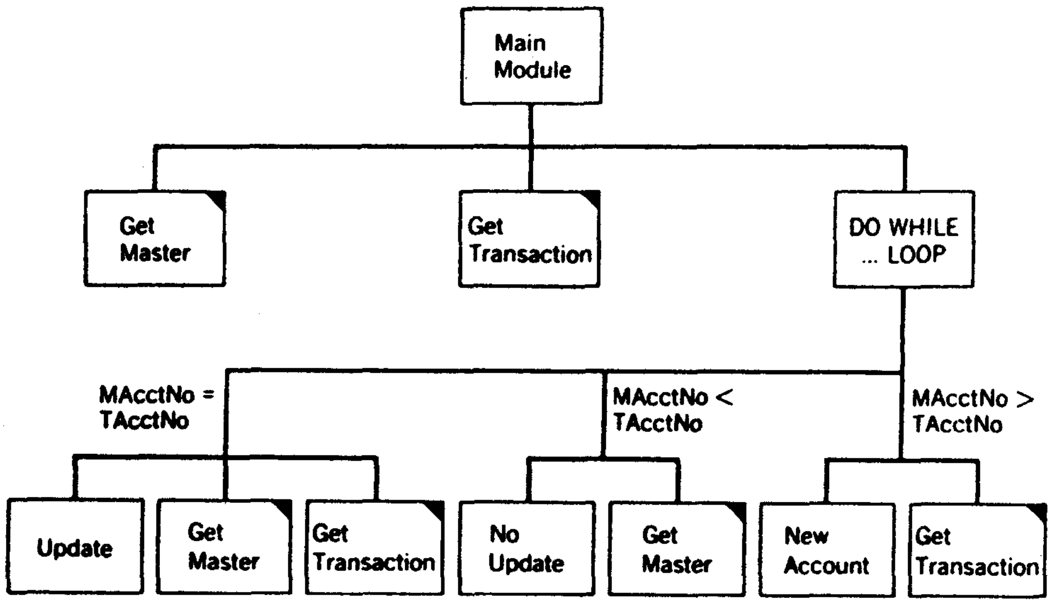
请记住:OldMstr 文件中的记录号是以 MAcctNo 排序的，而 TransIn 文件中的记录则是按 TAcctNo 来排序。 NewMstr 文件也将按账号顺序建立。

图 13.3 说明了在修改程序中各指令的伪代码。在这一修改程序中，采用了伪编码，使程序模块化，并且说明了程序是怎样自顶向下进行编码的。图 13.4 给出了层次图说明了各模块之间的关系。在读图 13.5 所示的程序之前,应仔细地研究这些框图。

伪代码

```
START
    打开文件
    从主文件中读数
    从细目文件中读数
    PERFORM 直至无其它输入?
        IF TAcctNo  =  MAcctNo
        THEN
            修改记录
            从主文件中读数
            从细目文件中读数
        ELSEIF MAcctNo  <  TAcctNo
        THEN
            不修改记录(从旧主文件中写一条新记录)
            从细目文件中读数
            ELSE
                增加一条新记录
                从细目文件中读数
            ENDIF
        ENDIF
    ENDPERFORM
    关闭文件
STOP
```

图 13.3　修改程序中的伪代码



注意：图中将 DO WHILE … LOOP 当作模块处理

图 13.4 修改程序中的层次图

```
'**********************************************
'This program updates an accounts receivable
'master file where multiple transactions per
'master are not permitted
'**********************************************
CLS
OPEN "OldMstr.dat" FOR INPUT AS #1
OPEN "TransIn.dat" FOR INPUT AS #2
OPEN "NewMstr.dat" FOR OUTPUT AS #3
CALL GetMaster (MAcctNo, MLastName$, MFirstName$, BalDue)
CALL GetTransaction (TAcctNo, TLastName$, TFirstName$, AmtOfSale)
DO UNTIL MAcctNo = 9999 AND TAcctNo = 9999
        IF MAcctNo = TAcctNo THEN
           CALL Update (MAcctNo,MLastName$,MFirstName$,BalDue,AmtOfSale)
           CALL GetMaster (MAcctNo,MLastName$,MFirstName$,BalDue)
           CALL GetTransaction (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        ELSEIF MAcctNo < TAcctNo THEN
           CALL NoUpdate (MAcctNo,MLastName$,MFirstName$,BalDue)
           CALL GetMaster (MAcctNo,MLastName$,MFirstName$,BalDue)
        ELSE
           CALL NewAccount (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
           CALL GetTransaction (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        END IF
LOOP
CLOSE #1,#2,#3
PRINT "END OF RUN"
END
'*****************************************************************
'     The Update procedure updates an existing
'        master record
'*****************************************************************
SUB  Update (MAcctNo,MLastName$,MFirstName$,BalDue,AmtOfSale)
               LET BalDue = BalDue + AmtOfSale
               WRITE #3, MAcctNo,MLastName$,MFirstName$,BalDue
END SUB
'*****************************************************************
'     The NoUpdate procedure re-creates a master record without
'        making any changes to it
'*****************************************************************
```

```
SUB  NoUpdate (MAcctNo,MLastName$,MFirstName$,BalDue)
        WRITE #3, MAcctNo,MLastName$,MFirstName$,BalDue
END SUB
'*********************************************************************
'     The NewAccount procedure creates a new master record
'        using data on the transaction file
'*********************************************************************
SUB   NewAccount(TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        WRITE #3, TAcctNo,TLastName$,TFirstName$,AmtOfSale
END SUB
'*********************************************************************
'     The GetMaster procedure reads a record from the master file
'         and determines if an end-of-file condition has occurred
'*********************************************************************
SUB   GetMaster (MAcctNo,MLastName$,MFirstName$,BalDue)
        IF EOF (1) THEN
                LET MAcctNo = 9999
        ELSE
                INPUT #1, MAcctNo,MLastName$,MFirstName$,BalDue
        END IF
END SUB
'*********************************************************************
'     The GetTransaction procedure reads a record from the
'        transaction file and determines if an end-of-file
'        condition has occurred
'*********************************************************************
SUB   GetTransaction (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        IF EOF (2) THEN
                LET TAcctNo = 9999
        ELSE
                INPUT #2, TAcctNo,TLastName$,TFirstName$,AmtOfSale
        END IF
END SUB
```

图 13.5(a) 用 Turbl Basic 编写的修改程序

OldMstr.Dat 文件

```
1,"NEWMAN","PAUL",100.5
2,"EVANS","LINDA",55
3,"MOUSE","MICKEY",44
4,"WASHINGTON","GEORGE",44.25
5,"MURPHY","EDDIE",35
```

TransIn.Dat 文件

```
1,"NEWMAN","PAUL",10
3,"MOUSE","MICKEY",50.5
4,"WASHINGTON","GEORGE",12
6,"SIMON","SIMPLE",30.5
```

NewMstr.Dat 文件

```
1,"NEWMAN","PAUL",110.5
2,"EVANS","LINDA",55
3,"MOUSE","MICKEY",94.5
4,"WASHINGTON","GEORGE",56.25
5,"MURPHY","EDDIE",35
6,"SIMON","SIMPLE",30.5
```

图 13.5(b) 数据文件

§ 13.2.3.3.1 主模块或主过程

首先，主模块打开两个文件．调用两个单独的过程，分别从主文件和细目文件中读取记录．文件号只用来跟踪文件． OldMstr 文件用#1，Tran5In 文件用#2，NewMstr 文件用#3．所以，一旦要打开、读或者关闭 OldMstr 文件，就必须用到文件号#1．

输入过程完成以后，主模块调用 Do UNTIL ... LOOP，使两个文件中的所有记录都被读入．当 MAcctNo 和 TAcctNo 的尾部数值为 9999 时，Do 为 UNTIL ... LOOP 结束．这些尾部数值是在各自的读入模块中赋给 AcctNo 的．

```
SUB GetMaster (MAcctNo, MLastName$, MFirstName$, BalDue)
    IF  EOF(1) THEN
            LET MAcctNo = 9999
    ELSE
            INPUT #1, MAcctNo, MLastName$, MFirstName$, BalDue
    END IF
END SUB
SUB GetTransaction (TAcctNo, TLastName$, TFirstName$, AmtOfSale)
    IF  EOF(2) THEN
            LET TAcctNo = 9999
    ELSE
            INPUT #2, TAcctNo, TLastName$, TFirstName$, AmtOfSale
    END IF
END SUB
```

在每个读模块中，除非遇到了文件结束条件，否则会一直读入记录．不论是主文件还是细目文件，当遇到文件结束条件时，各自的账号都赋上尾部数值 9999．而当两个文件的尾部数值均是 9999 时，DO UNTIL ... LOOP 将结束，在这里，我们假字 9999 不是有效账号，以便能用它作尾部值．

主循环结束后，关闭所有文件; 打印出工作结束信息; 执行过程也就结束了．

实际上在主模块之后，我们可采用自顶向下模块途经写出 Update，NoUpdate ,NewAccount，GetMaster 和 GetTransaction 的各个单独过程．在此加以说明是为了使用户能够明白输入文件是怎样处理的．在对自顶向下程序进行编码时，首先着重的是结构，其次才是填写这一过程的具体内容．

§13.2.3.3.2 在 DO UNTIL ... LOOP 中输入细目记录和主记录的处理方法

最初时，在主模块下执行 GetMaster 和 GetTransaction 这两个过程，从 OldMstr 文件和 TransIn 文件中得到一条记录。然后，用 DO UNTIL ... LOOP 将 OldMstr 文件中的账号 MAcctNo 和 TransIn 文件中的账号 TAcctNo 进行比较。由于这两个文件都是按各自的账号排序的，所以通过比较 MAcctNo 和 TAcctNo 即可确定下一个要执行一过程。比较 MAcctNo 和 TAcctNo，可能会发生三种情况：

(1) TAcctNo等于MAcctNo 这就意谓着在细目记录中存有与主文件中记录相同的账号。如果遇到这种情况，调用执行 Update 这一过程。修改 OldMstr 文件中的记录。在 Update 这一过程中，细目数据传送到主记录，而新的主记录则为以前的 BalDue 加上来自于细目记录的 AmtOfSale 得到。所以，OldMstr 文件的 4 个字段和细目文件中的 AmtOfsale 字段都必须传输到 Update 过程中去，然后，在 Getmaster 和 Gegtransaction 过程里，分别读人另一条来自于 OldMstr 文件 TransIn 文件的记录。而后，DO UNTIL ... LOOP 重复上述过程。

(2) TAcctNo大于MAcctNo

这表示主记录中的账号小于细目文件中的帐号。因为两个文件都是按账号排序的，所以这一条件意味着主记录存在而没有与互相对应的事物记录。这就是说：在当前修改周期内，主记录没有发生更变，而照原样写到 NewMstr 文件中。称这一过程为 NoUpdate。在 NoUpdate 过程中，利用旧主记录数据来写新主记录。所以，在这一过程中必须传送 OldMstr 字段，然后执行 Get Master 过程，来从 OldMstr 文件中读另一记录。由于 TAcctNo 比 OldMstr 文件中的 MAcctNo 大，我们没有处理这一细目记录。所以没有必要在 NoUpdate 过程中再去读另一细目记录。下例说明了如果 TAcctno 大于 MAcctNo 时所进行的处理：

MAcctNo	TAcctNo	处　理
0001	0001	修改主程序
0002	0003	0002 应原样写到 NewMstr 中；读下一条主记录，不必再读细目记录，因为 TAcctNo 为 0003 的细目记录还未处理。

(3) TAcctNo小于ActNo

由于两文件均是按账号排序的，所以，这一条件就意谓着细目记录存在而没有与

其相对应的主记录．根据执行修改过程的类型，这表示:或者是(1)处理来自于TransIn 文件中的一笔新账，或者是(2)出错;也就是说，TAcctNo 是错误的虽然在说明中，假设当 TAcctNo 小于 MactNo 时，这是一笔新账．但是，还是要考虑，如果 MAcctNo 大于 TAcctNo 时，所能执行的两种类型过程:

(A) 如果TAcctNo<MAcctNo，建立一笔新账．

如上所述，对于某些应用场合，若有细目记录而没有与之相对应的主记录则意味着是一笔新账，在我们的程序中称这一过程为 NewAccount．在这种情况，可以完全根据细目记录来建立一条新主记录，也就是说，我们用 WRITE#3，文件号#3 用于 NewMstr 文件中，将细目文件中的字段 TAcctNo、TLastName$、TFirstName$ 和 AmtOfSale 传送到这一过程中，而后执行 Get Transaltion 这一过程去读下一事务记录．此时，我们不必从 OldMstr 文件中去读另一记录，用为我们还未处理完大于 TAcctNo 的这条主记录．

(B) 如果TAcctNo < MAcctNo，表示出错

对某些应用场合，细目文件中的账号必须要和主记录中的账号对应相同;在这些应用中，新账号处理是采用其它不同的程序，而不属修改过程的一部分．

在这种情况下，如果TAcctNo小于MAcctNo，这将会引起一个错误过程的处理．在这一过程中，可以打上错误信息以标示出．通常，错误过程将打印出账号不匹配的细目记录．接着读下一条细目记录．

总之，从一单独的过程中读取主记录和细目记录，然后执行 DO UNTIL ... LOOP，在此比较账号大小，根据比较的结果，决定去执行 Update、NoUpdtae 还是 NewAccount 过程．DO UNTIL ... LOOP 将重复进行直到在 MAcctNo 和 TAcctNo 中出现了尾部数值为止，尾部数值表示这两个文件都达到了文件结束状态．

§13.2.3.3.3 举例说明修改过程

下面的例子，说明依据所读账号来决定运行的过程:

MAcctNo	TAcctNo	条件	处理过程
0001	0001	MAcctNo=TAcctNo	Update
0002	0004	MAcctNo<TActNo	NoUpdate
0003	0004	MAcctNo<TAcctNo	NoUpdate
0005	0004	MAcctNo>TAcctNo	NewAccount
0005	0005	MAcctNo=TAcctNo	Update

本节的最后将考虑如果一个文件较另一个文件先结束时的情形.

在这一修改程序中，假设对每条主记录来说只有单个细目记录. 在后面将考虑用同一账号的多个细目记录来修改单个主记录时所采用的过程.

再次回顾图 13.3 中的伪代码，图 13.4 中的层次图和图 13.5 中的程序.

§ 13.2.3.3.4 嵌套条件概述

修改程序的 DO UNTIL ... LOOP 中，当有三种条件需测试时，尤其使用一格式为 IF ... ELSEIF ... 的嵌套条件(nested Conditional). 回忆一下，嵌套条件的格式如下:

```
IF 条件 1 THEN
    语句组 1
ELSEIF 条件 2 THEN
    语句组 2
ELSE
    语句组 3
END IF
```

其意义为:

```
IF 条件 1 成立
    执行语句组 1
IF 条件 2 成立(条件 1 不成立)
    执行语句组 2
IF 条件 1 和条件 2 均不成立
    执行语句组 3
```

在程序中，下列内容被编码:

条　件	结　果
IF TAccNo = MAcctNo THEN	调用 Update
ELSEIF MAcctNo < FAcctNo THEN	调用 NoUpdate
ELSE	调用 NewAccount

如果 TAcctNo 小于 MAcctNo(同样 MAcctNo > TAcctNo) 执行最后 ELSE 的内容.

我们希望在对每一条主记录和细目记录处理时，能做为一个整体地执行 DO

UNTIL ... LOOP，那么最好采用嵌套条件结构，将其做为单个指令来编码.

§ 13.2.3.3.5 文件结束条件

有两个输入文件，就可能有二种文件结束条件发生. 首先，在结束文件 OldMstr 之前，文件 TransIn 可能已经结束，或者在 TransIn 文件结束之前已经执行完了 OldMstr 文件中的记录. 在程序中，这两种可能性都必须考虑到. 如果首先遇到 OldMstr 文件的结束条件，尾部数值 9'S 将存贮在 MAcctNo 中，由于 MAcctNo 中的 9999 不是"真值"或者不是有效数值，所以在说明中采用 9'S 来表示. 9'S 确保 TransIn 文件中余下的所有有效记录编号都小于 MAcctNo 中的尾部数值. 这样，TransIn 文件中的记录就可用来建立新主记录.

另一方面，如果首先遇到 TransIn 文件的结束条件，尾部数值 9'S 将存贮在 TAcctNo 中，这将确保 OldMstr 文件中所有剩余的有效记录按原样添加到 NewMstr 文件中.

只有当我们同时遇到二个文件的结束条件时，这才结束 DO UNTIL ... LOOP 过程. 当文件 OldMstr 和文件 TransIn 记录中的尾部数值均为 9'S 时，就会发生上述情况. 所以，能利用 TAcctNo 和 MAcctNo 中的尾部数值 9'S 来确保正确地处理二个文件中的所有剩余记录. 如果在二个文件中，数值 9999 是一有效数值，那么就要采用其它数用作尾部数值(如: 99999).

回忆一下，一读完上一条记录，就会遇到 EOF 条件. 在每条记录处理完后而不是在读时就需要测试一下这一条件. 所以在每次读之前，GetTransaction 过程都要测试 EOF. 当上一记录读人后，进行处理，此时，EOF 一直在测试，直到发出请求要读下一条记录为止.

假想这里只有两条记录，在循环之前将第一条在初始输入过程中读人，然后处理这条主记录. 在 DO UNTIL ... LOOP 中，将第二条主记录在 Get Master 过程中读人，因为在 GetMaster 过程执行第二次时，EOF 条件仍未遇到，所以读人了第二条记录或叫最后一条记录；而后才遇到了 EOF 条件，所以 EOF(1) 为真. 当第二次执行 GetMaster 过程时，将会遇到 EOF 条件，在那时，GetMaster 过程将给 MAcctNo 赋上尾部数值 9999，由于再也没有其它记录，所以输入文件中没有进一步读人的请求. TransIn 文件的处理完全与上述相同.

例 1: 先结束主文件

```
MAcctNo     TAcctNo     Action

9999        0007        New Account
 .           .           .
 .           .           .
 .           .           .
9999        9999        End of Loop
```

例 2: 先结束细目文件

MAcctNo	TAcctNo	Action
0008	9999	No Update
.	.	.
.	.	.
.	.	.
9999	9999	End of Loop

修改中过程的应用

注意到，修改程序中，采用过程:(1)执行相关的一组任务；(2)将被这些过程处理的数据在过程本身中传送。这使模块化程序中的数据和任务组都将结构化。

自我测验(一)

1. 关于专用应用程序的主要数据集叫作________.
2. 文件由________组成，第条________由叫________的数据单元组成.
3. 存贮顺序文件更变的文件叫________文件。
4. 在顺序修改过程，用了三种文件，分别为________、________、和________.
5. (T或F)在顺序文件的修改过程中，所有文件都必须按同一字段排序.
6. 在顺序的修改过程中，细目文件中的关键字段与________文件中的关键字段相比较.
7. 在第6题中，如果关键字段相等，则发生________过程，描述这一过程.
8. 在第6题中，如果细目文件中的关键字段大于主文件中的关键字段，将会发生________过程，描述这一过程.
9. 在第6题中，如果细目文件中的关键字段小于文件中的关键字段,将会发生________过程，描述这一过程.
10. 如果没有细目记录而仍有需处理的主记录，指出将要进行的处理过程.

§13.2.3.4 向顺序文件中添加记录

需要有一个向顺序文件的结尾处添写记录的单独过程．按如下方法打开文件:

```
OPEN "Master.Dat" FOR APPEND AS #1
```

用: WRLTE #1, ... 向数据文件 Master.Dat 结尾处添写记录．

§13.3 修改过程中的有效性检验

修改过程导致主文件发生了变化，所以，必须使错误减到绝对最小．为了"真实地"修改，必须应用大量的数据进行有效性验证，以便使错误减到最小．下面考虑某些常用的有效性检验程序．

§13.3.1 新账目的检验

用户将回想起，如果 TAcctNo 小于 MAcctNo，即存在细目记录而没有与之对应的主记录，则通常会用到两类过程．而在某些应用中，细目记录通常必须有与之对应的主记录；在这种情况下，如果 TAcctNo 小于 MAcctNo，就认为出错．

对其它的应用，如果 TAcctNo 小于 MAcctNo，细目记录则能成为一笔新账而添加到 NewMstr 文件中．但是,如果简单地将这一细目记录增加到主记录中，而没有任何附加的检验，这是非常不合适的．因为存在着 TAcctNo 编码不正确，实际上，细目记录不是一笔新账．通常在 TransIn 文件中，其记录本身需包含有一个编码字段．下面是 TransIn 文件中记录的一个完整格式:

TransIn 记录

TAcctNo, TLastName$, TFirstName$, AmtOfSale, Code

其中: Code = 1 表示 NewAccount;
Code = 2 表示 RegularUpdate;

TransIn 文件记录中的编码字段内为 1，表示此记录为一笔新账；为 2 则意味着这条记录是用来修改现存的主记录．

所以，如果 TAcctNo 小于 MAcctNo，只有当细目记录中编码字段为 1 时，才将这一细目记录当作一笔新账处理．而后，应修正 NewAccout 过程，以使输入的数据有效．

伪代码节选

```
IF TAcctNo < MAcctNo
    PERFORM
    IF 为新帐目
    THEN
        增加一条新记录
    ELSE
        写一个错误行
    ENDIF
  ENDPERFORM
ENDIF

PERFORM
    读取一条细目记录
ENDPERFORM
```

程序节选

```
SUB NewAccount(TAcctNo, TLastName$ TFirstName$, AmtOfSale, Code)
    IF Code = 1 THEN
        Write #3, TAcctNo, TLastName$, TFirstName$, AmtOfSale
    ELSE
        PRINT "Error in Code for Transaction Record"; TAcctNo
    ENDIF
END SUB

SUB GetTransaction(TAcctNo, TLastName$, TFirstName$, AmtOfSale, Code)
    IF EOF(2) THEN
        LET TAcctNo = 9999
    ELSE
        INPUT #2, TAcctNo, TLastName$, TFirstName$, AmtOfSale, Code
    ENDIF
END SUB
```

同样，Code 可用来验证在 Update 模块中处理的细目数据:

伪代码节选

```
PERFORM
    IF MAcctNo = TAcctNo
    THEN
        PERFORM
            IF 为正常调整
            THEN
                修改主记录
            ELSE
                写一个错误行
            ENDIF
        ENDPERFORM
    ENDIF
ENDPERFORM
```

程序节选

```
SUB Update(MAcctNo, MLastName$, MFirstName$, BalDue, AmtOfSale, Code)
    IF CODE = 2 THEN
        LET BalDue = BalDue + AmtOfSale
        Write #3, MAcctNo, MLastName$, MFirstName$, BalDue
    ELSE
        PRINT "Error in Code for Transaction Record"; MAcctNo
    END IF
END SUB
```

§13.3.2 双重检验关键字段

如果 MAcctNo 等于 TAcctNo，顾客姓名又相同的情况下，要有另一类有效性检验

用来确定修改程序的运行，这将会使传送错误账目细目的机会减到最小.

§13.3.3 关于从顺序主文件中删除编码和删险记录的检验

在以前的说明中，有一种修改功能没有考虑到，这就是主记录的删除。因为如果顾客放弃价格优惠或者没有付账，那么这笔账就要作费，就必须要用某些规定从主文件中消除这些特殊记录．在顺序文件的修改中，通过将记录不写进 NewMstr 文件中删除记录．用上面谈到过的编码处理字段技术来指定被删记录，可添加编码 3 指定要删除的记录．

TransRec 记录

TAcctNo, AmtOfSale, Code(1 — NewAccount, 2 — RegularUpdate,
3 — DeleteTheRecord)

编码为 1 表示一笔新账目；编码为 2 意谓着一般修改；编码为 3 则意味着相应的主记录将被删除.

可修正修改模块如下:

伪代码节选

```
PERFORM
    IF MAcctNo = TAcctNo THEN
        PERFORM
            IF 为正常调整
            THEN
                修改主记录
            ELSE IF 为删码除
            TNEN
                继续处理
            ELSE
                PERFORM
                    写一个错误行
                ENDPERFORM
            ENDIF
        ENDPERFORM
    ENDIF
ENDPERFORM
```

程序节选

```
SUB Update(MAcctNo, MLastName$, MFirstName$, BalDue, AmtOfSale, Code)
    IF CODE = 2 THEN
        LET BalDue = BalDue + AmtOfSale
        Write #3, MAcctNo, MLastName$, MFirstName$, BalDue
    ELSEIF Code = 3 THEN
    ELSE
        PRINT "Error in Code for Transaction # "; MAcctNo
    END IF
END SUB
```

注意，只有当 TAcctNo = MAcctNo 时，才执行修改过程；在执行过程以后，从每个文件中读另外一条记录．如果 Code=3，OldMstr 文件中对应记录将不写到新主文件中去；实际上是通过不处理或者不再写进新主文件来删除这条记录．当遇到这一条件时，可编码 ELSEIF Code =3，Then ELSE ... 来简单地跳过被删除记录．在修改过程中，Code 为 2 或者 3 时，是不正确的．

§ 13.3.4 顺序错误的检验

在一修改过程中，细目文件和主文件中记录的顺序是很关键的；如果在主文件或者细目文件中有一条或多条记录的顺序弄错了，整个运行将产生错误的结果．所以在修改程序之前或者在修改程序中，必须要删除这些错误．考虑如下：

	TransIn TAcctNo	TransIn TAcctNo	
	0006	0006	
	0009	0009	
序号错误的记录 →	0118	0114	
	0015	0015	由于 TransIn 中第 0118 号记录顺序出错，所以这些记录都被错误处理．
	0016	0016	
	⋮	0017	
	⋮	⋮	

如果存有顺序错误的可能，则在程序中包括顺序检验是必要的．首先建立一个叫 CheckSeq 的存储变量如下：

```
Checkseq=0
```

然后，用每条细目记录确认后一记录的账号是否大于前一条记录的帐号:

```
SUB GetTransaction(TAcctNo, TLastName$, TFirstName$, AmtOfSale, Code)
    IF EOF(2) THEN
        LET TAcctNo = 9999
    ELSE
        INPUT #2, TAcctNo, TLastName$, TFirstName$, AmtOfSale, Code
    ENDIF
    IF TAcctNo < CheckSeq THEN
        CALL SeqError(TAcctNo)
        EXIT SUB
    ELSE
        CheckSeq = TAcctNo
    END IF
END SUB
```

第一次执行 GetTransaction 过程时，TAcctNo 将大于 CHeckseq。这是因为 CHeckseq 开始时为零；以后，如果比较结果为 TAcctNo"高于"它应该的值，那么用 TAcctNo 替换以前的值。

在 SeqError 过程中，用户将要:(1)简单地打印信息，指出所发生的顺序错误；(2)停止运行；或者(3)计算出顺序错误的数目，如果计算值超过了预先设定的界限，就中止运行。

由计算机排序的文件不必进行顺序检验

中止运行，重新排序细目记录，而后再修改主记录，这将是一个冗长且费用高的过程。有一较好的方法，就是在处理之前，在磁盘上建立一个 TransIn 文件，利用排序实用程序，由计算机对其进行排序。经计算机排序后，就能够确保被排序过的文件是严格地按顺序的，在这种情况下，就没有必要再有顺序检验过程。

用户能使用 DOS 系统中的 SORT 指令对文件进行简单排序，这又快且容易，但不是万能的。

```
A> SORT < 要分类的文件 > 结果文件
```

例

```
A> SORT < Unsorted.Trn > TransIn
```

在这类 SORT 中，通常假设是按每条记录的第一个字段进行排序的，在说明中，因为要按每条记录中的第一个字段 TAcctNo 来排序，所以能采用这类排序。如果不是按第

一个字段排序或者第一个字段的长度对每条记录不相同，就不能使用这类排序。此处，DOS 中的 SORT 不能处理大于 63K 字符的文件。还必须记住，利用 DOS 系统执行的 SORT 是按 ASCII 码排序的，认为小写字母大于大写字母，而大写字母又大于数字。

还有一些其它的排序程序或者应用程序，如 QSORT 和 Info-Sort。它们对其它的应用可能是通用。

Borland International 公司的 Turbo Data Base Toolbox 中也有排序程序。

§13.4 为各主记录修改带多个细目记录的过程

迄今为止，我们将重点都集中在采用单个细目记录来改变主记录内容的修改过程。对某些应用程序，只需要单个事物记录就够了。例如，在 Sales 文件中，对当前周期，只需要包含有销售员的销售总数的单个细目记录，这一记录将用来修改与他或者她有关的主记录。

而对其它的应用，在修改周期内，可能需要处理不止一个更变。例如：修改一个应收取账目主文件，可能要用到多个细目记录，如若在当前修改周期中，一个顾客要买 12 项，那么对每一个顾客主记录将有 12 条细目记录，这就需要有另一不同类型的修改过程。

图 13.5 中描述的修改过程，只有在每条主记录允许一条细目记录的情况下才适用。如果不止有一个细目记录和主记录有相同的账号，那么将不能够正确地处理第二条细目，因为 TransIn 文件和 OldMstr 文件中记录的关键字段相等，这一条件会引起向 NewMstr 文件写记录和从 TransIn 文件和 OldMstr 文件读记录。这样的处理过程将是不恰当的。

图 13.6 给出了修改文件的伪代码，在此过程中每条主记录允许有多个处理。

所以，如果每条主记录可能有多个细目记录，则需要如图 13.7 所示的 Turbo Basic 程序；其层次图见图 13.8。

```
START
     Open the files
     PERFORM
          Read a master record
     ENDPERFORM
     PERFORM
          Read a transaction record
     ENDPERFORM
     PERFORM until no more input
          IF MAcctNo = TAcctNo
          THEN
               +--------------------------------------------------+
               |PERFORM                                           |
               |     DO WHILE MAcctNo = TAcctNo                   |
               |          Add transaction amount to the master    |
               |          Read a transaction record               |
               |     ENDDO                                        |
               |     Write a new master record                    |
               |     Read a master record                         |
               |ENDPERFORM                                        |
               +--------------------------------------------------+
          ELSEIF  MAcctNo < TAcctNo
          THEN
               Write a new master record from the old master
               Read a master record
               ELSE
                    Write a new master from the transaction
                    Read a transaction record
               ENDIF
```

```
            ENDIF
        ENDPERFORM
        PERFORM
            End-of-job operations
        ENDPERFORM
    STOP
```

图 13.6 修改文件的伪代码，文件中各主记录允许有多条细目记录

```
'*********************************************
'This program updates an accounts receivable
'master file where multiple transactions per
'master are permitted
'*********************************************
CLS
OPEN "OldMstr.dat" FOR INPUT AS #1
OPEN "MultTran.dat" FOR INPUT AS #2
OPEN "NewMstr.dat" FOR OUTPUT AS #3
CALL GetMaster (MAcctNo, MLastName$, MFirstName$, BalDue)
CALL GetTransaction (TAcctNo, TLastName$, TFirstName$, AmtOfSale)
DO UNTIL MAcctNo = 9999 AND TAcctNo = 9999
        IF MAcctNo = TAcctNo THEN
                CALL Update (MAcctNo,MLastName$,MFirstName$,
                        BalDue,TAcctNo,TLastName$, TFirstName$,AmtOfSale)
                CALL GetMaster (MAcctNo,MLastName$,MFirstName$,BalDue)

        ELSEIF MAcctNo < TAcctNo THEN
                CALL NoUpdate (MAcctNo,MLastName$,MFirstName$,BalDue)
                CALL GetMaster (MAcctNo,MLastName$,MFirstName$,BalDue)
        ELSE
            CALL NewAccount (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
            CALL GetTransaction (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        END IF
LOOP
CLOSE #1,#2,#3
PRINT "END OF RUN"
END
'******************************************************************
'     The Update procedure updates the master record
'       assuming that there may be multiple transactions
'       for a master
'******************************************************************
SUB  Update (MAcctNo,MLastName$,MFirstName$,BalDue,
                TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        DO WHILE MAcctNo = TAcctNo
                CALL MultTransaction (TAcctNo,TLastName$, TFirstName$,_
                        AmtOfSale,BalDue)
        LOOP
        WRITE #3, MAcctNo,MLastName$,MFirstName$,BalDue
END SUB
'******************************************************************
'     The NoUpdate procedure is executed if there is no
'       transaction record for a master; in this case
'       the old master is written onto the new master file as is
'******************************************************************
SUB  NoUpdate (MAcctNo,MLastName$,MFirstName$,BalDue)
        WRITE #3, MAcctNo,MLastName$,MFirstName$,BalDue
END SUB
'******************************************************************
'     The NewAccount procedure is executed if there is a
'       transaction record with no corresponding master;
'       in this case, a new master is created from the
'       transaction record
'******************************************************************
SUB  NewAccount (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        WRITE #3, TAcctNo,TLastName$,TFirstName$,AmtOfSale
END SUB
'******************************************************************
'     The GetMaster procedure reads a master record and tests for
'       an end-of-file condition
'******************************************************************
```

```
SUB   GetMaster (MAcctNo,MLastName$,MFirstName$,BalDue)
        IF EOF (1) THEN
              LET MAcctNo = 9999
        ELSE
              INPUT #1, MAcctNo,MLastName$,MFirstName$,BalDue
        END IF
END SUB
'*********************************************************************
        The GetTransaction procedure reads a transaction record
        and tests for an end-of-file condition
'*********************************************************************
SUB   GetTransaction (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
        IF EOF (2) THEN
              LET TAcctNo = 9999
        ELSE
              INPUT #2, TAcctNo,TLastName$,TFirstName$,AmtOfSale
        END IF

END SUB

SUB MultTransaction  (TAcctNo,TLastName$, TFirstName$,AmtOfSale, BalDue)
              LET BalDue = BalDue + AmtOfSale
              CALL GetTransaction (TAcctNo,TLastName$,TFirstName$,AmtOfSale)
END SUB
```

图 13.7(a) 每条主记录有多条细目记录的 Turbo Basic 程序

OldMstr.Dat 文件

```
1,"NEWMAN","PAUL",100.5
2,"EVANS","LINDA",55
3,"MOUSE","MICKEY",44
4,"WASHINGTON","GEORGE",44.25
5,"MURPHY","EDDIE",35
```

MultTran.Dat 文件

```
1,"NEWMAN","PAUL",10
1,"NEWMAN","PAUL",20
1,"NEWMAN","PAUL",30
3,"MOUSE","MICKEY",50.5
4,"WASHINGTON","GEORGE",12
4,"WASHINGTON","GEORGE",10
6,"SIMON","SIMPLE",30.5
```

NewMstr.Dat 文件

```
1,"NEWMAN","PAUL",160.5
2,"EVANS","LINDA",55
3,"MOUSE","MICKEY",94.5
4,"WASHINGTON","GEORGE",66.25
5,"MURPHY","EDDIE",35
6,"SIMON","SIMPLE",30.5
```

图 13.7(b) 数据文件

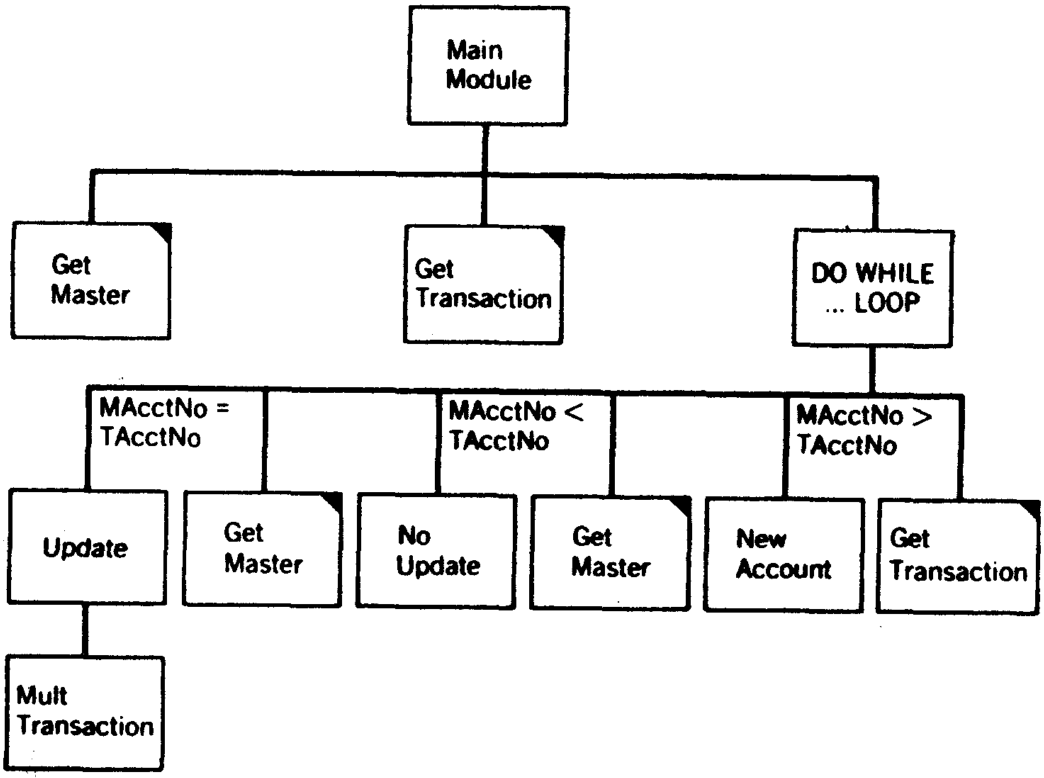


图 13.8　每条主记录有多条细目记录的层次图

在调试程序时显示输出文件

当调试程序，将磁盘用作输出时，用户应该检验输出记录以保证这些记录是正确的。通过现有的 Turbo Basic，在 DOS 系统下，执行这一程序就可做到，演示如下：

```
A> TYPE  文件名
```

正如以前所提到的，对数据文件尤其要包括.dat 的扩展名。

或者，仅仅为了在屏幕上或在打印机上查看记录，用户可打印(LPRINT)记录或显示(PRINT)记录。

自我测验(二)

1. 为检验细目记录是否用于建立一笔新账目，在细目记录中应含有________。
2. (T或F)对每条主记录要处理不止一条细目记录，这是不可能的。
3. (T或F)当细目记录的关键字段与主记录中的不相匹配时，细目记录是错误的。
4. 要察看数据文件中的内容，可用DOS系统命令________。

5. 为了确认文件是按顺序的，在修改过程应用该文件之前，必须________。
6. 编写一段程序，用来建立一个主文件。文件中包括有社会保险号(关键字段)，学生姓名，已修完的学分或学时数。
7. 编写一段程序，用细目文件中的输入记录来修改题6中的主文件，细目文件中包含有社会保险号、学生姓名、和这学期所修的学分或学时数。
8. 编写一段程序，向题6中的文件添加记录。
9. (T或F)通过细目文件中包有从编码字段来识别要删除的记录，这是可行的。
10. (T或F)为删除主文件中的记录，可以不将它写入或者重建新主文件。

§13.5 随机文件结构

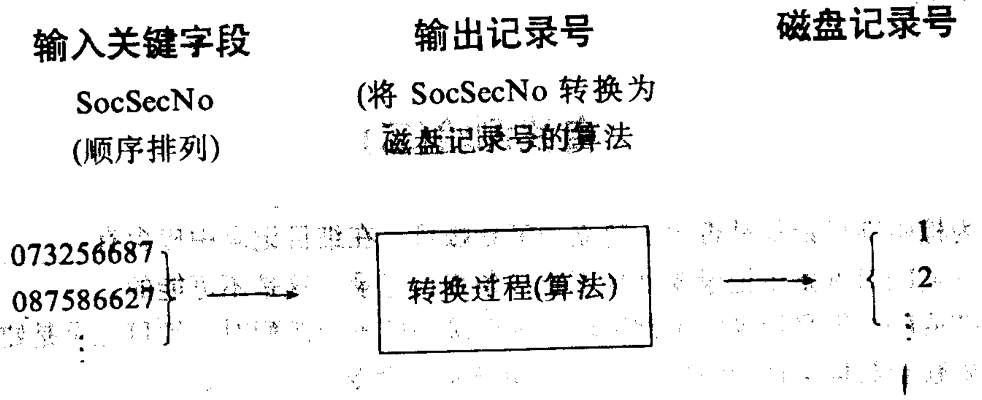
§13.5.1 顺序文件与随机文件中的区别

随机文件或者称直接存取文件是这样构成的，文件中的记录能够直接检索，而不需要按照顺序去读每条记录直到找到所需要的记录为止。假设有一随机主文件，有主账号我们叫它 MAcctNo，且将它作为关键字段，还假设 MAcctNo 的范围是从 0001 到 9875。若要寻找一条 MAcctNo 为 0987 的记录，计算机简单地运行到文件中第 987 条记录位置，而不需要读完前 986 条记录。

所以，构造磁盘中随机存取记录的最简单方法是采用关键字段，其取值范围为 1 到 n，其中关键字段应完全连续(1,2…n)。按照这一方法，关键字段会指出记录的位置，在此位置就能找到相对应的记录(如：关键字段为 883 的记录就处在磁盘的第 883 条记录位置上)。

但是，关键字段与记录位置不直接对应，即假设我们有一工资名单文件，只有 SocSecNo 和 EmpName $ 可做为关键字段，但两个中任意一个字段都不能用来直接指明记录位置，社会保险号太长而且根本不连续，因为在我们的文件中可能有 1000 个社会保险号，其值从 001027738 到 986421179。EmpName $ 是另一个有问题的关键字段，因为它不是数值型的。且除此之外，它不是唯一的(很多人可能有同一名字)。在这种情况下，要建立一个能随机存取的文件，要通过利用算法或公式将关键字段转换成记录位置。当要从这个文件中读时，也需用同样的算法寻找或检索这一记录。

例



在读 SocSecNo 为 087586627 这条记录时，用到和建立这一文件所用到的相同的转换过程；这一转换过程将指出所需的记录在文件中是第 2 条。

实际上，是通过说明被用作记录编号的关键字段来开始我们对随机文件讨论。例如：关键字段 AcctNo，其值从 001 到 989 能够用来作记录编号。在下一节中，将解释非连续关键字段转换成相关联的记录编号的转换过程。用在建立文件时的转换过程与用在存取或修改文件中的是相同的。

§13.5.2 处理顺序文件和随机文件中的区别

§13.5.2.1 顺序文件中的记录长度可变化 而随机文件中的记录长度是固定的

用 Turbo Basic，顺序文件在长度上是绝对可变化的，因为每个字段都是用逗号相互隔开的。相反，随机文件的长度是固定的。也就是说：顺序文件是由记录组成，而每条记录则为一字段符串，每个字段则是根据需要由许多字符组成。因为顺序文件中的字段是以逗号相隔开的，所以在这类文件被称作“带字段定界符顺序文件”。

带字段定界符的顺序文件记录举例：概述

	姓名	地址	工资
1.	"Paul NewMan",	"125 Main St, NYC, NY 10036",	97000
2.	"Linda Evans",	"463 Midwood Ave, Mailbu, CA 11758",	214575

因此，带定界符顺序文件中的字段是由逗号相互隔离开的，而每个字段只用记录所指明的字符个数。因为记录长度不是常量而是根据数据来确定的，所以，称这些记录长度可变(Variable length)。

而随机文件则需要每条记录中的每一字段里存贮有相同数目的字符，因此，随机文件是由固定长度记录组成的。

在随机文件中，固定长度记录将浪费磁盘空间，因为既使其每单个字段实际占用的位置要小的多，但每一字段还均必须扩大到最大值。因此，如果有一个人的名字是 20 个字符，则所有的人名字字段都要有 20 个字符，虽然有的记录需要较少的位置就可以了。如果数据占用的位置比留用的位置少，则随机文件中的字段将用低位或最右边的空格来填充；或者，也可用高位或最左边的空格来填充。这由程序员指定。

随机文件中的记录举例

姓 名 (20 个字符)	地 址 (45 个字符)	工资 (六位整数)
"Paul NewMan"	"125 Main St, NYC, NY 10036"	97000
"Linda Evans"	"463 Midwood Ave, Mailbu, CA 11758"	214575

其中: 姓名和地址两字段均为高位对齐，低位添空；工资字段为低位对齐，高位添空；

应注意到，随机文件中记录没有用逗号来分隔各字段，也没有用引号来指明是字符串。因为在预先已指明了每一字段的长度和类型，所以就不需要分界符了.

总之，顺序文件有长度可变化的优点，而随机文件必须是长度固定的。

§13.5.2.2 随机文件占磁盘空间少且处理速度比顺序文件快

随机文件中记录如用和顺序文件中记录相同的位置数目来表示数据，将要比这些数据在顺序文件中占用的磁盘空间少。这是因为随机文件不需要逗号或者引号分界符。此外，随机文件的处理要比顺序文件快.

§13.5.2.3 随机文件常不可移植

注意，随机文件通常是不可移植的(Transportable)。这就意味着用户可能不能够利用文本编辑程序来“读”或打印它们。进一步说，用 Turbo Basic 程序建立的随机文件不能够被用其它语言编写的程序或者用不同的 Basic 编辑程序编写的程序采用。相反，顺序文件常可移植到大多数程序中，甚至可移植到数据库管理程序中.

何时建立顺序文件，何时建立随机文件

当记录只需按顺序存取时或者对每个应用其大多数或者全部记录需处理时，最好建立顺序文件; 但是如果记录的访问或读取是随机的，或者只有很少量的记录只需处理一次，此时最好选择建立随机文件.

§13.5.3 随机文件处理

§13.5.3.1 建立随机文件

用于建立随机文件的指令如下:

§13.5.3.1.1 打开一个将用作输出的随机文件

用 OPEN 启动准备要处理的文件

```
OPEN 文件说明 AS [#]文件号 LEN = 记录长度
```

参数 LEN 指定了文件为随机文件．正如我们已经指出的，随机文件和顺序文件，是有固定的记录长度的。用户必须在 OPEN 语句中的 LEN 分句里指出每一记录的长度．随机文件不能象顺序文件用 FOR INPUT 或 FOR OUNPUT 语句来打开，这是因为随机文件在同一程序中既可读又可写．所以在同一程序中，不管是输入还是输出，所有的随机文件均可用．

例:

```
OPEN "Employee.Dat" AS #1 LEN = 80
```

这条 OPEN 语句可在磁盘上建立一个名叫 Employee.dat 的文件．文件名必须根据以前所指出的规则来确定一文件名需 1 到 8 个字符、字母、数字和某些特殊字符．文件名的开头必须是字母，其后可以跟有一可任意选择的由三个字符组成的扩展名，扩展名与文件名之间用句号分开。建议在数据文件中采用扩展名.dat 以区分 Turbo Basic 程序文件。Turbo Basic 程序文件通过 Trubo Basic 编译程序加.Bas 的扩展名．

OPEN 语句的更明确格式，编码如下:

```
OPEN "Employee.Dat" FOR OUTPUT AS #1 LEN = 80
```

在上例中，#1 是文件号．在此程序中，应用这个文件的所有其它输入和输出指令都必须用这一相同的文件号．在程序中，文件号仅仅是识别文件的通道．

总之，对随机文件，OPEN 语句指明了文件名和文件号，还指出了文件中各记录的长度，注意随机文件没有象在顺序文件中那样指明了输出(OUTPUT)或输入(INPUT)，但它指明了用于随机 FOR RANDOM．即，正如我们以后可见到的，从一个随机文件中读或向此文件中写都是用同一个 OPEN 语句．

§ 13.5.3.1.2 定义输出到随机文件的字段

对随机文件，FIELD 语句紧跟在 OPEN 语句之后，指出了记录中的字段长度和字段名; 随机文件中的所有字段都将按字符串变量处理，任何数值型数据如关键字段和／或算术运算结果，如果要将它输出到随机文件中字段中，都将被转换成字符串变量．要采用预先定义好的特殊函数来将数值变量转换成随机文件中的字段．这就是说，简单的赋值语句，不能用于在随机文件中将数据传送到字段中去．首先考虑一下用来描述随机文件中字段的 FLELD 语句．其格式为:

```
FIELD 文件号 字段长度 AS 字符串变量, [字段长度 AS 字符串变量] …
```

文件号必须和 OPEN 语句中所用的相同，这样计算机才知道 FIELD 语句中涉及到哪个文件．

例

```
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1, 4 AS EmpNo$, 25 AS EmpName$, 25 AS Address1$, _
          4 AS Salary$, 20 AS JobDescription$, _
          2 AS DeptNo$
```

随机文件 Employee.dat 将有如下固定长度记录格式:

EmpNo$	EmpName$	Address1$	Salary$	Job Description$	DeptNo$
1　　4	5　　29	30　　54	55　　58	59　　78	79　　80

输出到随机文件中字段是输入变量或者是处理结果，在任何一种情况中，这些变量都必须用预先定义的特殊函数将其转换成字符串变量。

§ 13.5.3.2　用 RSET 和 LSET 给随机文件中的字段传送数据

§ 13.5.3.2.1　LSET 和 RSET 语句

正如以前所提到的，不能简单地采用赋值语句将数据移到随机文件的字段中。如果传送给字段的数据已经是字符串格式，可以用 LSET 和 RSET 中的任何一个函数向字段传送数据;' 如果要传送的数据不是字符串格式，必须先将其转换，然后才用 LSET 或者 RSET 函数向字段传送数据。

LSET 指令左对齐将数据传送到字段。这就是说，数据串传送到字段的最左边位置，其它没有填满的最右边位置则用空格来代替，例如：将"P. smith"采用 LSET 指令传送到长度为 10 的 EmpName $ 字段中，将变成"P. Smith　　"，同样用 LSET 指令将"123"传送到长度为 6 的字段中将为"123　　"

函数 RSET 同样用于在随机文件中向字段传送数据串，但数据在字段中是右对齐而不是左对齐，即用 RSET 指令将"P. Smith"传送到长度为 10 的字段中将变成"　　P. smith"，而"123"传送到长度为 6 字段中时将会成为"　　123"。

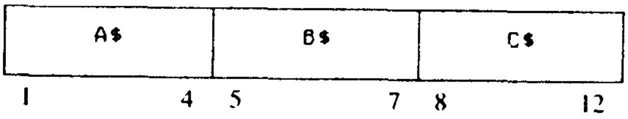
LSET 和 RSET 的格式如下:

$\left.\begin{matrix} LSET \\ RSET \end{matrix}\right\}$　字段变量 ＝ 字符串表达式

注意，如果传送给字段变量的数据和字段长度是相等的，或者传送的数据要长些，用 LSET 和 RSET 将产生同样的结果。即用 LSET 或者 RSET 向字段传送"ABCD"得到 ABC，其中"D"被截取了。只是在传送数据比接收字段长度短时，LSET 和 RSET 之间才有差别。如果是这样的话，LSET 则左对齐数据，而 RSET 则右对齐数据。

例:

我们将在随机文件中用 LSET 来向下面字段传送字符串变量.



考虑下面 FIELD 和 LSET 指令:

```
FIELD #1, 4 AS A$, 3 AS B$, 5 AS C$
LSET A$ = "ABCD"
LSET B$ = "ABCD"
LSET C$ = "ABCD"
```

下面的数值将存放在这三个字段中:

用 LSET 语句传送字符串变量

字段名	字段长度	存储值	解　释
A$	4	ABCD	A$ 4 位字段，因此它将文字“ABCD”原样存放
B$	3	ABC	B$ 存放“ABCD”最左边的三个字符，截去最右边字符
C$	5	ABCDb̸	C$ 为 5 位字段; 它存放带低位或最右边空格的“ABCD”

§ 13.5.3.2.2　在随机文件中用 RSET 代替 LSET 向字段传送字符串变量

```
FIELD #1, 4 AS A$, 3 AS B$, 5 AS C$
RSET A$ = "ABCD"
RSET B$ = "ABCD"
RSET C$ = "ABCD"
```

下面的值将被存贮在这三个字段中:

用 RSET 语句传送字符串变量			
字段名	字段长度	存储值	解　释
A $	4	ABCD	A $ 4 位字段，因此它将文字"ABCD"原样存放
B $	3	ABC	与 LSET 一样，B $ 存放"ABCD"最左边的三个字符，截去最右边的字符。
C $	5	b̸ABCD	C $ 为五位字段，存放带高位或最左边空格的"ABCD"。

注意，不论用 LSET 还是用 RSET 向字段传送数值，若数值包含的字符数目多于指定的字段长度，如在上述两个例中的 B $ 字段，则最右边的字符被截取，例如用 LSET 或者 RSET 将"ABCDE"向一 4 位置段传送，其结果将是 ABCD 存入了字段。

利用 LSET 和 RSET 的编码规则

这里有二个由程序员使用的约定:

1. 用LSET向字段传送字符串数据，而用RSET向字段传送数字型数据。通常采用这一约定是因为数字型数据通常在字段中右对齐(常为b̸b̸123，而不是 123b̸b̸)，而字符串数据在字段中常为左对齐(为MAXWELLb̸b̸，而不是 b̸b̸MAXWELL)
2. 统一用LSET或者统一用RSET。不考虑字段中数据的类型。如果用一致的方法向字段传送数据，那么当我们从随机文件中读时，出错的机会少些。为了进一步处理需将字段转换成字符串变量。

§ 13.5.3.2.3　将数据变量转换成字符串字段

正如以前所提到的，随机文件中所有字段均必须定义成字符串变量，任何随机记录中的数字首先需转换成串；而后才能用 LSET 或者 RSET 函数传送到字段中去。将数值型转化成字符串格式过程是通过采用 "Make"函数来完成的。用"make"函数将数值型数据转换成字符串格式，其目的是为随机文件的输出做准备。下面是"make"函数的例子:

字符串变量 = MKI$(整型表达式)
字符串变量 = MKL$(长整型表达式)
字符串变量 = MKS$(单精度表达式)
字符串变量 = MKD$(双精度表达式)

注意，这些 make 函数返回的是字符串数据，因此用一个美元号($)作为此函数名的一部分。

如果数值变量是按标准单精度形式输入的，或者被赋给了一单精度形式的值，我们将采用 MKS$ "make"函数，在我们的程序中大部分数值变量就是单精度值。

可以将"make"函数与 LSET 或 RSET 指令联合起来，先将数值函数据转化成串形式，而后传给字段。例如：

LSET *RSET* }	字段 = MKS$(单精度表达式)

例：

RSET Salary$ = MKI$(SalaryIn%)
RSET DeptNog$ = MKI$(DeptNo%)

确定存贮数值型数据字段的长度

随机文件中包含有数值型数据的字段可以是 2、4 或者是 8 字节的。一个字节即为一个存储位置。2、4 或者 8 字节字段的精度是依靠字段所包含的数值型数据的类型而定。

例如，包含整型数据字段必须是 2 字节长，整型数据范围从−32768 到 32767，这是用 2 字节存贮时−，二进制记数法表示的范围。所以，如从 01 到 50 的一组数是一整型变量，那么它可以用 2 字节存贮器存贮在随机文件中的字段里。

假设，我们希望输入一组数字，并且将输入的值做为我们建立的随机文件中的字段。内容如下：

```
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1 ...
     2 AS DeptNo$   'DeptNo$ must be defined as 2 bytes
     :
     :
INPUT DeptNoIn%    'Inputted variable is in integer form
RSET DeptNo$ = MKI$(DeptNoIn%)
     :
     :
```

这些指令：

(1) 建立了一个2字节字段DeptNo$，此字段为随机文件记录的一部分；

(2) 输入名为DeptNoIn%的整型变量;

(3) 将整型数值变量DeptNoIn%转换成随机文件所需的字符串格式，且在字段中右对齐数据；这就意谓着，字段中没有填上的高位位置将用空格代替。

有些数值型变量，如在最初说明中的雇员编号，可能包含有大于整型数上限 32767 的整型数值，那么这些数值必须按长整型数输入。类似地，有些数值变量，如工资，可能需要小数位数，这些需按单精度数输入。长整型数和单精度数两者都用 4 字节存储在随机文件的字段中。

例

```
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1, 4 AS Salary$
   ⋮
' Salary is entered as a single-precision number
```

```
INPUT SalaryIn
LSET Salary$ = MKL$(SalaryIn)
```

或者希望按长整型数据输入 Salary:

```
INPUT SalaryIn&    ' Salary is entered as a long integer
LSET Salary$ = MKL$(SalaryIn&)
```

不管在哪一种情况，Salary$ 中所包含的值是单精度数还是长整型数，均必须在 FIELD 语句用 4 字节或 4 个存储位置来定义。

所以，我们可以按单精度数输入 SalaryIn，然后转换成合适的字符串格式以便它们能存贮在随机文件的字段中；或者我们可以输入一个长整型数 SalaryIn $，而后将其转换成 Salary $，这是假设被输入的工资没有小数位数,即只是按整元输入的。

类似地，如果雇员编号包含有大于 32767 的数值，其也必须按单精度数或者长整型数输入(变量名称后跟有 $)。在这种情况下，假设雇员编号只为整型数值，所以，更适合用长整型数形式:

```
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1, 4 AS EmpNo$    'EmpNo$为四字节字段
   ⋮
```

```
INPUT EmpNoIn&
LSET EmpNo$ = MKL$(EmpNoIn&)
```

在这种情况下，输入 EmpNoIn $ 且存贮在随机文件中一 4 字节表字段里，用合适的"make"函数来完成转。

双精度数用 8 字节来存贮。一个字段包含有大于 10^{38} 的数值，这个字段就建立成 8 字节字段，按双精度数读入，并转换成适当的形式。

```
FIELD #1 ··· 8 AS Num$
INPUT Num#
LSET Num$ = MKD$(Num#)
```

下表总结了随机文件中字段的转换规则:

输入变量类　型	举例	数值范围	随机文件中FIELD 语句中的字段长度	将数字变量转化为随机文件中字段所需的"Make"函数
整型	DeptNoIn%	-32768～32767	2	MKI $
长整型	EmpNoIn&	-2^{31}～$2^{31}-1$	4	MKL $
单精度型	SalaryIn	10^{-38}～10^{38} (包括小数值)	4	MKS $
双精度型	Num#	10^{-308}～10^{308} (包括小数)	8	MKD $

带有字符串数据的字段，其长度是根据字符数来定的，所以由 20 个字母组成的雇员姓名字段的定义如下:

```
FIELD #1 20 AS EmployeeName$
```

自我测验(三)

1. 顺序文件记录有________长度格式，而随机文件记录有________长度格式。
2. 记录中字段由逗号分隔开的文件是________文件。
3. (T或F)随机文件既可随机访问，又可顺序访问。
4. 比较且讨论顺序文件和随机文件的可移植性。
5. 在________语中的________参数指明了一个文件为随机文件。
6. (T或F)当顺序文件被打开以后，用户能在同一程序中读或写它。
7. (T或F)在同一程序中随机文件既可作为输入用又可作为输出用。

8. 在一随机文件中，采用OPEN和FIELD语句编码包含有下列字段.

StudentId%	5位(整型)
StudentName$	20个字符
UnitsComplete%	3位(整型)
GPA%	3位(整型)

9. 写出将第8题中StudentId%，UnitsComplete%和GPA转换成字符串字段的语句.
10. 对题8和题9中的字段，用LSET和RSET语句传送数据.

以后，将着重点集中于从随机文件中读和修改随机文件. 记住: 如果要从随机文件中读出一个字段且利用算术运算来处理它，那么首先必须要用"make"函数的逆函数将其转换成数值.

用于随机文件中将数值变量转换成字符串字段的Make函数	相对应的将随机文件中字段转换成数值格式的函数
MKI$	CVI
MKL$	CVI
MKS$	CVS
MKD$	CVD

不要用简单的赋值语句给字段赋值

利用"make"函数，考虑下列赋值语句:

无效编码(Invalid Coding)

```
FIELD #1, 4 AS Salary$
S$ = MKS$(Salary)            ' 这是一条赋值语句
```

最后这条指令将一叫 Salary 的数值变量转换成字符串变量; 但是，B$ 也许不能被定义成随机文件中的字段. 因为只有利用 LSET 或 RSET 函数，才能给随机文件中字段赋上字符串数据.

虽然下面需要附加编码，但是这是正确的.

有效编码(Valid Coding)

```
FIELD #1, 4 AS Salary$
```

```
        ⋮
S$ = MKS$(Salary)          ' 这是一条赋值语句
LSET Salary$ = S$
```

正如我们将看到的，大多数例子用的都是单精度数；单精度数对典型的应用将是足够大的。记住，随机文件中包含有单精度数的字段，在它们的转换形式中需要4字节存贮单元。

不要将"make"函数的STR$函数相混淆。STR$是将函数值转换成可打印串形式，而不能用于将数字转换到随机文件中的字段里。

注意即使是作为字符串变量输入的数据也需要用LSET或RSET将其转换到随机文件中的字段里。但是，这里字段的长度是根据要存贮的实际字符数来确定的。

§13.5.3.2.4 向随机文件写记录

要建立一个随机文件，将输入数据，必要时还需按某些方法去处理它，用LSET和RSET函数将其传送到输出字段，然后，我们用PUT语句将记录写到一随机存取文件中。

PUT指令的格式如下：

```
PUT  文件号 [, 记录号]
```

PUT指令将向指定的文件号写记录，文件号必须是与OPEN语句和FIELD语句中的一样。每条记录都将按放在记录号指定的记录位置上，记录号就是记录编号。注意：我们用的是PUT语句而不是WRITE语句来建立随机文件。

记录编号本身是可以任意选择的。如果省略了，则Turbo Basic将在磁盘文件中的下一个顺序空间写记录，所以输出的第一条记录在磁盘上就是记录1，第2条将为记录2，等等。虽然记录编号不是需要的，但是我们建议用户通常要包括它。这样可使程序更加清楚。

§13.5.3.2.5 关闭随机文件

在将所有的记录都已写入随机文件以后，必须用CLOSE语句关闭文件，其格式是：

```
CLOSE  [文件号]
```

在此，文件号就必须再一次和用在OPEN、FIELD及PUT指令中的相同。如果在CLOSE语句中设有指明文件号，则将关闭所有的文件。

例:

假设希望向随机文件号写记录. 首先输入一变量用来表示随机文件中记录编号, 然后向每条记录输入信息. 当信息输入时, 可按同样的顺序将信息写到文件中, 可用下列程序按顺序将记录写到一随机文件中:

```
CLS
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1, 4 AS EmpNo$; 25 AS EmpName$, 25 AS Address1$,
          4 AS Salary$, 20 AS JobDescription$, _
          2 AS DeptNo$
PRINT "Enter Number of Records:"
INPUT NoOfRec%
FOR I% = 1 TO NoOfRec%
    INPUT EmpNoIn
    INPUT EmpNameIn$
    INPUT Address1In$
    INPUT SalaryIn
    INPUT JobDescriptionIn$
    INPUT DeptNoIn%
    LSET EmpNo$ = MKS$(EmpNoIn)      Convert to string
    LSET EmpName$ = EmpNameIn$
    LSET Address1$ = Address1In$
    LSET Salary$ = MKS$(SalaryIn)    Convert to string
    LSET JobDescription$ = JobDescriptionIn$
    LSET DeptNo$ = MKI$(DeptNoIn%)
    PUT #1, I%
NEXT I%
CLOSE #1
PRINT "End of Job"
END
```

用关键字段作为相对记录号向随机文件写记录

以后为了随机地从上述文件中检索或者读记录, 需要知道被读的每条记录的记录编号. 即, 指导计算机去读记录#3 和#7, 等等, 这是指需要跟踪每条记录的编号, 最好采用关键字段. 它既可以被用作记录数本身, 又可转换成记录编号. 例如, 如果雇员编号是从 1 变化到 n, 其中 n 表示任意数值, 可以用 EmpNoIn 作为记录编号.

下列是一段将 EmpNoIn 用作记录编号或关键字段的程序节选. 每个 EmpNoIn 不需要按顺序输入, 一条 EmpNoIn 为 16 的记录将写到第 16 个记录位置; 一条 EmpNoIn 为 456 的记录将写到第 456 个位置位置, 等等. 用这种方法, 以后, 如果我们知道了希望存取记录的 EmpNo, 就可以从文件中随机地读取. 如果输入一 EmpNoIn 为 43, 指导计算机去存取这一记录, 它将走到第 43 个磁盘记录位置去读这条记录. 采用 9999 作为尾部数值以标示工作的结果.

下面为文件创建程序:

```
OPEN "Employee.Dat" AS #1, LEN = 80
FIELD #1 4 AS EmpNo$, 25 AS EmpName$, 25 AS Addressl$,
         4 AS Salary$, 20 AS JobDescription$, _
         2 AS DeptNo$
PRINT "Enter Data--Type -9999 For Emp No When Done"'
INPUT EmpNoIn
    DO WHILE EmpNoIn    -9999 '-9999 is a trailer value
    INPUT EmpNameIn$
    INPUT AddresslIn$
    INPUT SalaryIn
    INPUT JobDescriptionIn$
    INPUT DeptNoIn%
    LSET EmpNo$ = MKS$(EmpNoIn)    'Convert to string
    LSET EmpName$ = EmpNameIn$
    LSET Addressl$ = AddresslIn$
    LSET Salary$ = MKS$(SalaryIn)  Convert to string
    LSET JobDescription$ = JobDescriptionIn$'
    LSET DeptNo$ = MKI$(DeptNoIn%)
    PUT #1, EmpNoIn
    INPUT EmpNoIn
LOOP
CLOSE #1
PRINT "End of Job"
END
```

按这种方法，利用 EmpNoIn 作为记录编号将每条记录写到磁盘随机文件中. EmpNoIn 不必按顺序输入，也不必是连续的。如果键入终端的第一个 EmpNoIn 是 006，那么，记录将被按放在磁盘文件中的第 6 个记录位置上，如果键入终端的下一个 EmpNoIn 为 053，那么第 53 条记录将被按放在磁盘上。例如，若没有 EmpNoIn 为 05 或没有 EmpNoIn 为 153，那么，第 5 条或者第 153 条记录位置将是空的。在所有的雇员数据都输入后，输入 9999 表示工作结果.

自我测验(四)

1. (T或F) STR $ 和MKS $ 函数可以相互交换，用来将数据转换到随机文件的字段中.
2. (T或F) LSET和RSET函数只能用于将数值域转换到随机文件的字段中。
3. (T或F) 在随机文件中，记录编号可任意选择.
4. 为了向随机文件写记录，需采用________语句.
5. (T或F)利用关键字段来指明文件中记录的相对位置，且用来向随机文件写记录，这是可能的.
6. 如果一个非连续性关键字被用来向随机文件写记录，会发生什么情况?
7. 向随机文件的字段中转换一个单精度数时，需要________字节存贮单元.
8. 向随机文件的字段中转换一个整型数时，需要________字节存贮单元.
9. 从下列输入数据，编写一段程序，建立一随机文件.

InventoryNoIn%	4 位的整数
ItemDescIn $	20 个字符

NoOnHandIn%　　　　　　3 位的整数
ItemPriceIn　　　　　　5 位数字

10. 修改题9中的编码，以便可按关键字段InventoryNo $ 写记录.

§ 13.5.3.3　从随机文件中读：概述

§ 13.5.3.3.1　为查询进行随机存取

从一随机文件中既可随机地又可顺序地读记录，尤其是当进行查询时，此程序能够使经理对那些需要查询的雇员的记录进行检索. 在这种情况下，请求并不一定是按顺序的，即:经理可能首先请求是关于 EmpNo 为 227 的信息，而后是 EmpNo 为 115 的，等等.

§ 13.5.3.3.2　为制作报表进行顺序存取

也可以从随机文件中顺序地读记录，尤其是在用作报表目的时. 也就是说，当产生一个详细报表或总结报表或者甚至是异常报表时，通常我们按顺序读整个文件.

不论随机地还是顺序地从随机文件中读记录，都可用 Get 指令. GET 语句的格式和 PUT 语句类似.

GET　文件号，记录号

如用 PUT 语句一样，文件号可以任选. 如果省略了，Turbo Basic 将按顺序读每一记录. 注意我们用 GET 而不 INPUT 语句从随机文件中存取记录.

§ 13.5.3.3.3　关于修改的随机存取

同样，通过随机读入记录来修改随机文件. 也就是说，雇员更变不必按顺序输入 . 例如，可以先是 EmpNo 为 0887 的记录姓名更变了，紧跟着的是 EmpNo 为 116 的记录工资增加了. 在这种情况下，随机地从随机文件中存取或读记录而后将更变直接写到同一记录上. 注意,当打开一个随机文件,时，可以根据需要，在同一程序上进行读和写,这是因为对随机文件 OPEN 语句不需指明是 INPUT 还是 OUTPUT.

§ 13.5.3.3.4　必须将从随机文件中读取的字符串数据转换为可用的数值形式

请回忆一下，在随机文件中，数值和字符串数据两者均必须用 LSET 或 RSET 指令

向字段里传送. 此外, 数值变量必须用"make"函数将其转换成随机文件所需要的适当形式。类似地, 作为字符串数据存贮在随机文件中的数, 当其作为输入读取时, 必须将它转换为相应的数值形式, 才能够使用.

数值变量类型	建立随机文件时, 将数字转换成字符串 在磁盘文件中建立记录所用的 Make 函数	为使用从输入随机文件中按字符串读入的数字, 而进行的相应转换
单精度(如: N)	MKS $	CVS
双精度(如: N#)	MKD $	CVD
整型数(如: N%)	MKI $	CVI
长整型(如: N&)	MKL $	CVL

例

打印雇员编号为 5 的工资, 编码如下:

```
GET  #1, 5        ' 获取第 5 号记录
PRINT "Salary is: "; CVS(Salary$)
```

假定建立文件时, 工资是按单精度数输入的, 且采用 MKS $ 函数将其转换成串格式, 所以可用 CVS 将 Salary $ 转换成单精度数.

§ 13.5.3.4 为了查询随机读取随机文件

假想, 已拥有用雇员编号建立的雇员记录文件, 雇员编号指明了每条记录在文件中的位置(参见 § 13.5.2.5 节).

随机文件中每条记录均按雇员编号顺序排列. 即文件中第一条记录的雇员编号为"01", 下一条为"02", 等等. 接着写一段查询程序, 按任意顺序输入任何一组雇员编号, 然后从随机文件中打印出与之相对应的雇员姓名和工资.

```
CLS
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1, 4 AS EmpNo$, 25 AS EmpName$, 25 AS Address1$,
          4 AS Salary$, 20 AS JobDescription$, _
          2 AS DeptNo$
PRINT "Enter Employee Number - Type -9999 when done"
INPUT EmpNoIn
DO  WHILE EmpNoIn <> -9999
    GET #1, EmpNoIn
    PRINT EmpName$; " has a salary of "; CVS(Salary$)
```

```
    INPUT EmpNoIn
LOOP
CLOSE #1
PRINT "End of Job"
END
```

此程序从雇员数据文件中随机地读记录。如果 EmpNo 为 04 是我们查询的第一条记录，那么在文件中第 4 条记录就会被直接读入，而不必读完前三条记录。

这一存取技术可用在有关键字段的任何时候，诸如雇员编号，其值范围从 01 到 × ×，其中 × × 是任何数字。当写随机文件时,雇员编将用来指明记录位置。在现在查询程序中，雇员编号被用来指明记录所处的位置。也就是说，雇员编号为 587 的记录就处在第 587 个记录位置。

所以，当建立一随机文件时，可用每条记录的关键字段(Key field)来指明相对应记录在文件中的位置。库存文件以零件号作为关键字段，应收取账目文件以顾客编号的关键字段，等等。如果没有这类关键字，但它们或多或少从 1 连续变化到 n，那么必须将每条记录的关键字段用一公式转换化成记录编号，在这章的结尾，将概述一下某些转换技术。

注意，在上述的例子中，每一输入的 EmpNoIn 指明了要存取的记录编号，在文件上必须有一相对应的 EmpNo$。如果存在有输入的 EmpNoIn 与文件上的 EmpNo$ 字段不相匹配这种可能性的话，那么将编出一段错误控制规程。在这以后出现更多的错误。

§ 13.5.3.5 顺序读取随机文件

正如以前所提到的，也可以从随机文件中顺序地读。而对查询，如上述，典型地是从文件中随机地读，但用于产生报表的程序通常要从文件顺序地读。

回想一下，当从文件中顺序地读时，必须一直读到遇着文件结束条件时为止。采用 EOF 函数来处理顺序文件。用来处理顺序文件的一种方法是：

```
AreThereMoreRecords$ = "Yes"
CALL ReadModule(    )        'Parameters included here
DO  WHILE AreThereMoreRecords$ = "Yes"
     .  {
     .  { Process
     .  {

    CALL ReadModule (   )
LOOP
.
.
.
END
```

```
SUB ReadModule (   )
   IF  NOT EOF(1)  THEN    'EOF is an end-of-file indicator
        INPUT #1 ...
   ELSE
        AreThereMoreRecords$ = "No"
   END IF
END SUB
```

EOF 函数，只能用在顺序文件中，而不能用在随机文件中。对随机文件，可采用文件长度函数(LOF)确定随机文件所用的字节数或者存贮位置数。为了确定在随机文件中的实际记录数，可用字节总数(LOF)除以记录长度。记录长度在 OPEN 语句中指明了。如果记录长度为 80，那么 LOF(1)／80 将表示在文件#1 中的记录数。处理 LOF(1)／80 的分法与处理 EOF(1)相同。

因此，当要从随机文件中顺序地读所有记录，且打印出每一雇员姓名时，编码如下：

```
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1 4 AS EmpNo$, 25 AS EmpName$, 25 AS Address1$_
        4 AS Salary$, 20 AS JobDescription$_
        2 AS DeptNo$
Rec% = 1
DO  WHILE Rec% <= LOF(1) / 80
    GET #1, Rec%
    PRINT EmpName$
    Rec% = Rec% + 1
LOOP
```

或者利用一个初始读和一个叫做 ReadModule 过程。可编码如下：

```
AreThereMoreRecords$ = "Yes"
Rec% = 1
CALL ReadModule(Rec%, EmpName$, AreThereMoreRecords$)
DO  WHILE AreThereMoreRecords$ = "Yes"
    PRINT EmpName$
    Rec% = Rec% + 1
    CALL ReadModule (Rec%, EmpName$, AreThereMoreRecords$)
LOOP
END
SUB ReadModule (Rec%, EmpName$, AreThereMoreRecords$)
    IF  Rec% <= LOF(1) / 80 THEN
        GET #1, Rec%
    ELSE
        AreThereMoreRecords$ = "No"

    END IF
END SUB
```

测试 Rec%以确定它是否小于或等于 LOF(1)与记录长度之商。用 LOF 函数重复执行循环，直到文件号为#1 的文件被全部读完为止。

§13.5.3.6　随机地修改随机文件

当修改一个随机文件时，可随机地读取文件中的记录，且用新的数据修改或替换；即，在同一程序中可以读、或者写磁盘随机文件。对照作为输入读取的雇员编号修改工资，其程序如下：

```
CLS
OPEN "Employee.Dat" AS #1 LEN = 80
FIELD #1, 4 AS EmpNo$, 25 AS EmpName$, 25 AS Address1$, _
          4 AS Salary$, 20 AS JobDescription$, _
          2 AS DeptNo$
PRINT "Enter Employee Number-Type -9999 When Done"
INPUT EmpNoIn
DO   WHILE EmpNoIn <> -9999
     GET #1, EmpNoIn
     PRINT "Enter New Salary"
     INPUT SalaryIn
     LSET Salary$ = MKS$(SalaryIn)
     PUT #1, EmpNoIn
     INPUT EmpNoIn
LOOP
CLOSE #1
PRINT "End of Job"
END
```

上述程序对所有输入的雇员编号进行了修改，这是一个修改的例子．其它类型的修改有:

1. 既允许向随机文件增加记录又允许更变记录的修改．
   为了完成这一要求，用户必须指出将指定的雇员编号是看成新雇员，还是现有雇员．

```
CLS
OPEN "Emp2.Dat" AS #1 LEN = 80
FIELD #1, 4 AS EmpNo$, 25 AS EmpName$, 25 AS Address1$, _
       4 AS Salary$, 20 AS JobDescription$, _
       2 AS DeptNo$
CALL EnterData (EmpNoIn, New$)
DO WHILE EmpNoIn <> -9999
       IF  New$ = "N" THEN
           CALL UpdateRoutine (EmpNoIn, EmpNo$, EmpName$, _
                Address1$, Salary$, JobDescription$, DeptNo$)
       ELSEIF New$ = "Y" THEN
           CALL NewEmployeeRoutine (EmpNoIn, EmpNo$, EmpName$, _
                Address1$, Salary$, JobDescription$, DeptNo$)
       ELSE
           PRINT "Error in input"
       END IF
       CALL EnterData (EmpNoIn, New$)
LOOP
CLOSE #1
Print "End of Job"
END
'*****************************************************************
' This procedure inputs an Employee Number and determines whether
' the employee is new or not
'*****************************************************************
SUB EnterData (EmpNoIn, New$)
    Print "Enter Employee Number -- Type -9999 When Done"
    Input EmpNoIn
```

```
    Print "Is this a new employee? (Y or N)?"
    Input New$
END SUB
'*****************************************************************
' This procedure updates the salary for each employee number
' entered as input where the user specified that the employee
' was not new
'*****************************************************************
SUB UpdateRoutine(EmpNoIn, EmpNo$, EmpName$, Address1$, Salary$, _
        JobDescription$, DeptNo$)
    GET #1, EmpNoIn
    PRINT "Enter New Salary"
    INPUT SalaryIn
    LSET Salary$ = MKS$(SalaryIn)
    PUT #1, EmpNoIn
END SUB
'*****************************************************************
' This procedure creates a record on the file for all new
' employees
'*****************************************************************
SUB NewEmployeeRoutine (EmpNoIn, EmpNo$, EmpName$, _
        Address1$, Salary$, JobDescription$, DeptNo$)
    PRINT "Enter Employee Name: "
    INPUT EmpNameIn$
    PRINT "Enter Address: "
    INPUT Address1In$
    PRINT "Enter Salary: "
    INPUT SalaryIn
    PRINT "Enter Job Description: "
    INPUT JobDescriptionIn$
    PRINT "Enter Department Number: "
    INPUT DeptNoIn%
    LSET EmpNo$ = MKS$(EmpNoIn)
    LSET EmpName$ = EmpNameIn$
    LSET Address1$ = Address1In$
    LSET Salary$ = MKS$(SalaryIn)
    LSET JobDescription$ = JobDescriptionIn$
    LSET DeptNo$ = MKI$(DeptNoIn%)
    PUT #1, EmpNoIn
END SUB
```

2. **既允许工资变化又允许其它变化的修改**

为了完成这一要求，用户必须指出:对输入的每一雇员编号，都需要进行什么类型的更变，下面的节选说明了这一点:

```
SUB UpdateProcedure
    PRINT "Enter 1 FOR Salary Change"
    PRINT "Enter 2 FOR Name Change"
    PRINT "Enter 3 FOR Address Change"
    PRINT "Enter 4 FOR Job Description Change"
    PRINT "Enter 5 FOR Dept Change"
    INPUT TypeOfChange
    SELECT CASE TypeOfChange
    CASE 1
        CALL SalaryChange
    CASE 2
        CALL NameChange
    CASE 3
        CALL AddressChange
```

```
    CASE 4
        CALL JobChange
    CASE 5
        CALL DeptChange
    CASE ELSE
        CALL ErrorModule
    END SELECT
END SUB
```

3. 标出删除记录的修改

为了完成这一要求，每一记录必须包含有一活动状态码．对活动记录，有一数值；对不活动记录，有一不同的数值。该数值用来标明要删除的记录．这一修改可改变活动码．

例: 下列程序建立了一带有活动状态码字段的文件:

```
CLS
OPEN "Emp3.Dat"  AS #1 LEN = 81
FIELD #1, 1 AS ActiveFlag$,4 AS EmpNo$, 25 AS EmpName$, 25 AS Address1$,
        4 AS Salary$, 20 AS JobDescription$,
        2 AS DeptNo$
PRINT "Enter Employee Number "
INPUT EmpNoIn
DO WHILE EmpNoIN <> -9999
        INPUT EmpNameIn$
        INPUT Address1In$
        INPUT SalaryIn
        INPUT JobDescriptionIn$
        INPUT DeptNoIn%
        LSET ActiveFlag$ = "A"
        LSET EmpNo$ = MKS$(EmpNoIn)   'Convert to string
        LSET EmpName$ = EmpNameIn$
        LSET Address1$ = Address1In$
        LSET Salary$ = MKS$(SalaryIn)
        LSET JobDescription$ = JobDescriptionIn$
        LSET DeptNo$ = MKI$(DeptNoIn%)
        PUT #1, EmpNoIn
        INPUT EmpNoIn
LOOP
'************************************************************
'This FOR ...NEXT loop prints out the file for verification
'purposes
'************************************************************
FOR I% = 1 TO LOF(1)/81
        GET #1, I%
        EmpNoX = CVS(EmpNo$)
        IF EmpNoX = I% THEN
                LPRINT Activeflag$; CVS(EmpNo$); EmpName$; Address1$;
                LPRINT
                LPRINT CVS(Salary$); JobDescription$; CVI(DeptNo$)
                LPRINT
                LPRINT
                LPRINT
        END IF
NEXT I%
CLOSE #1
PRINT "End of Job"
END
```

创建的文件

```
A 1 PAUL NEWMAN              111 MAIN SI NYC
 30000 CLERKA               2

A 3 LINDA EVANS              333 MADISON AVE NYC
 50000 SUPERVISOR           1

A 7 NANCY STERN              555 FIFTH AVE NYC
 80000 DP MANAGER           2
```

下列程序向文件增加或者从文件中删除记录:

```
CLS
OPEN "Emp3.Dat" AS #1 LEN = 81
'*******************************************************
' Note that this employee file has one added position
' at the beginning of each record designated as an
' ActiveFlag$.  This one position field will contain a
' value of A for active records and a value of I for
' inactive records.  When the file is created initially
' the first position of each record must be set to A.
'*******************************************************
FIELD #1, 1 AS ActiveFlag$,4 AS EmpNo$, 25 AS EmpName$,
          25 AS Address1$, 4 AS Salary$, 20 AS JobDescription$,
           2 AS DeptNo$
CALL EnterData (EmpNoIn, Reply$)
DO WHILE EmpNoIn <> -9999
        IF Reply$ = "D" THEN
                CALL DeleteRoutine (EmpNoIn,ActiveFlag$, EmpNo$,
                        EmpName$, Address1$, Salary$,
                        JobDescription$, DeptNo$)
        ELSEIF Reply$ = "N" THEN
                CALL NewEmployeeRoutine (EmpNoIn,ActiveFlag$, EmpNo$,
                        EmpName$, Address1$, Salary$,
                        JobDescription$, DeptNo$)
        ELSE
                PRINT "Error in input"
        END IF
        CALL EnterData (EmpNoIn, Reply$)
LOOP
'*******************************************************
  This FOR ... NEXT loop prints out the names and
  activity status code for all the records
  in the file for verification purposes.
'*******************************************************
FOR I% = 1 to LOF(1)/81
        GET #1, I%
        EmpNoX = CVS (EmpNo$)
        IF EMPNoX = I% THEN
                PRINT EmpName$, ActiveFlag$
        END IF
NEXT I%
CLOSE #1
PRINT "End of Job"
END
```

```
'*************************************************************
' This procedure inputs an Employee Number and determines
' whether the record with that corresponding number is
' to be deleted from or added to the file
'*************************************************************
SUB EnterData (EmpNoIn, Reply$)
        PRINT "Enter Employee Number -- Type -9999 When Done"
        Input EmpNoIn
        PRINT "Type N for New or D for Delete (or done)"
        Input Reply$
END SUB
'*************************************************************
' This procedure deletes records from a file by deactivating
' them.  That is, the ActiveFlag$ field is changed for
' records that are to be deactivated
'*************************************************************
SUB DeleteRoutine (EmpNoIn,ActiveFlag$, EmpNo$,
                   EmpName$, Address1$, Salary$,
                   JobDescription$, DeptNo$)
        GET #1, EmpNoIn
        PRINT "Employee Number ";CVS(EmpNo$);"is to be deleted
        PRINT "Are you sure?  Enter Y or N"
        INPUT Check$
        IF Check$ = "Y" THEN
                LSET ActiveFlag$ = "I"
                LPRINT "Employee Number"; CVS(EmpNo$);"has been deleted"
        END IF
        PUT #1, EmpNoIn
END SUB
'*************************************************************
' This procedure adds new employee records to the random file
'*************************************************************
SUB NewEmployeeRoutine  (EmpNoIn,ActiveFlag$, EmpNo$,
                   EmpName$, Address1$, Salary$,
                   JobDescription$, DeptNo$)
        PRINT "Enter Employee Name: "
        INPUT EmpNameIn$
        PRINT "Enter Address: "
        INPUT Address1In$
        PRINT "Enter Salary: "
        INPUT SalaryIn$
        PRINT "Enter Job Description: "
        INPUT JobDescriptionIn$
        PRINT "Enter Department Number: "
        INPUT DeptNoIn%
        LSET EmpNo$ = MKS$(EmpNoIn)
        LSET EmpName$ = EmpNameIn$
        LSET Address1$ = Address1In$
        LSET Salary$ = MKS$(SalaryIn)
        LSET JobDescription$ = JobDescriptionIn$
        LSET DeptNo$ = MKI$(DeptNoIn%)
        LSET ActiveFlag$ = "A"
        PUT #1, EmpNoIn
        LPRINT "Employee Number";CVS(EmpNO$); " has been added
               to the file"
END SUB
```

打印结果

Employee Number 1 has been deleted

Employee Number 4 has been added to the file

运行后的 Emp3.Dat 文件

The following are names and activity statuses

of all records on the file

PAUL NEWMAN	I
LINDA EVANS	A
DONALD DUCR	A
NANCY STERN	A

注意，如果允许对文件进行删除，则在建立此文件时必须有活动状态字段；活动状态字段用来指定已建立的每条记录的"活动"。当需删除这一记录时，给此码赋不同的值。其它程序用于从随机文件中顺序地读时，正处理记录之前，必须确定记录是活动还是不活动。

注意，当用作报表目的从文件中顺序地读时，每条 GET 应紧跟一测试语句，以确定这一记录是活动还是不活动，然后相应地处理它。

自我测验(五)

1. (T或F)随机文件的记录既可随机地读又可顺序地读。
2. (T或F)在同一程序中，可以读或者写随机文件。
3. 用________语句从随机文件中检索记录。
4. (T或F)被作为字符串数据存贮在随机文件中的数字，在应用它之前，必须将其转换回数值形式。
5. 用________函数来测试随机文件的结束。
6. (T或F)从随机文件中删除的记录是物理删除。
7. 编一段程序，从定义如下的随机文件中顺序地读和打印所有记录。

```
FIELD #1, 2 AS InventoryNo$, 20 AS iTEMdESC$,_
          2 AS NoOnHand$, 4 AS ItemPrice$, 1 AS ActiveFlag$
```

   InventoryNo$ 为关键字段。
8. 编一段程序，从如第7题的定义文件中随机地读和打印所有记录。
9. 编一段程序，从如第7题定义的文件中随机地删记录。假设在此文件中建立一活动状态字段，其值为"A"表示活动记录。通过改变此字段中的值来删除记录。
10. 编一段程序，从定义如题7的文件中随机地添加记录。

§13.5.3.7 处理输入/输出文件错误

在处理磁盘文件时，可能有各种类型的输入/输出(I/O)错误，其中最常见的 I/O 错误为:

OPEN	a. 找不到指定的文件. b. 文件已经打开.
GET PUT	指定的文件长度正确.
GET 文件号，记录号	此文件不存在.
PUT 文件号，记录号	要写的记录已在文件中.

对所有的输入／输出错误，可以写一错误处理过程. 这一过程可用来指出错误发生在哪里，要采取什么措施；如果适当的话，指出在哪一点应继续执行下去.

Turbo Basic 中采用 GOTO 语句，提供的错误处理程序如下:

```
    ⋮
GET #1, RecNoIn%
ON ERROR GOTO ErrorHandler
    ⋮
PUT #1, RecNoIn%
ON ERROR GOTO ErrorHandler
    ⋮
```

ErrorHandler 过程包括如下指令:

指令	含　义
PRINT ERDEV, ERDEV$	打印设备信息
PRINT ERL, ERR	在出错处打印一条线和错误类型
RESUME NEXT	即使发生了输入／输出错误，程序仍继续执行一下条指令.
RESUME Lalel	恢复标签. 使程序能在不同处恢复. 例如: RESUME module X，表示在模块中的输入／输出错误标上 Module X 后，计算机继续执行.
RESUME 0	RESUME 0 能使程序返回到出错语句处——只有在错误处理过程中，含有一固定的错误情况下，才采用这一指令.

在此，简单地提出了一些错误处理过程的要求，而没有详细地解释它们. 因为:

(1) 它们需要使用GOTO，而GOTO使程序非结构化；

(2) 它们较复杂；

(3) 大多数公司都有它们自己的特殊要求.

我们建议采用用户自己的错误测试，以确保建立的记录有效，保证正确随机地访问记录和恰当地进行修改．最好的方法是，在屏幕上回送用户所有的结果和在用户每次运行结束时打印跟踪记录．同样,如果需随机地访问记录，在处理之前必须进行检验．

例: 查看作为输入读入的雇员编号和姓名与文件中的是否相匹配?

```
INPUT EmpNoIn%, NameIn$
GET #1, EmpNoIn%
EmpNo% = CVI(EmpNo$)
IF  EmpNoIn% = EmpNo% AND EmpName$ = NameIn$ THEN
    CALL ProcessRecord ( ) 'Parameter list goes in ( )
ELSE
    PRINT "Error"
END IF
```

§ 13.5.3.8 其它文件处理语句：概述

§ 13.5.3.8.1 KILL 语句

有时在开始处理之前，用户可能要删除一个文件，用 KILL 语句可达到这一目的．

```
KILL  "文件名"
```

这一语句尤其用于当程序用来建立随机文件，且用户要确保同名的文件已经不存在时．

例

```
KILL "Inventory.Dat"
OPEN "Inventory.Dat" AS #1, LEN = 80
```

§ 13.5.3.8.2 Name 语句

用 Name 语句给文件换名，正如以前所提到的，在顺序文件中将文件 New Mstr 更变成文件 OldMstr 的编码是:

```
NAME  "文件名 1" AS "文件名 2"
```

例

```
NAME "NewMstr.Dat" AS "OldMstr.Dat"
OPEN "OldMstr.Dat" FOR INPUT AS #1
```

§ 13.5.3.9 关于将关键字转化成记录编号的算法

如前所述，不是所有文件都有能用作记录编号的关键字段．例如：在工资文件中的社会保险号关键字，就不能用作记录号．同样地，如果细目文件的细目编号关键字，它从 1 变化到 500，又从 1500 变化到 2700，从 9000 变到 11000 等等，它也不能有效地用作记录号．

在这种情况下，必须将关键字段转化成记录编号．将关键字段转化成记录编号所用的方法叫作散列"(Hashing)"．

散列算法包括如下：

随机化或散列算法

下面将一个数字关键字转换成相关的记录编号．在此，用 AcctNo 作关键字段，但任何其它数值关键字段均能采用．

算法	解 释	举 例
折半 (Folding)	将字段分割成两部分或更多部分，然后相加；(根据文件大小) 截取多余的数字．	1. AcctNo＝0125 a. 分割和相加： 01+25 记录将为 26 b. 此记录将放在第26个磁盘位置 2. AcctNo＝2341 分割和相加，23+41＝64 为记录编号
数字提取	在固定的数字位置提取数字，在选择数字位置之前应试着分析数字分布	1. Acct＝0125，如果假设第 2 个和第 4 个数量是最均匀分布，将定记录编号为 15． 2. 记录编号为31，某值可能是从关键字段 AcctNo＝2341 中提取的．

平方值截取	将关键字平方后，截取所需的数字数目	1. AcctNo＝0125 a. 平方后得值15625 b. 截取3 位，625作为记录编号．

散列组合方法举例

1. 关键字InventoryNo的值＝384756
   a. 分割成364+756＝1140
   b. 平方1140^2＝1299600
   c. 截取4位数字
   d. 记录编号为9600
2. 关键字SocialSecurityNumber的值＝547362845
   a. 分割547+362+845＝1745
   b. 平方1745^2＝3076516
   c. 截取4位数字
   d. 然后得记录编号为6516

冲突（Couisions）

当通过随机化算法，两个或更多字段被传送到用一个物理记录编号时，就会发生冲突．当建立或者访问随机文件，其中的关键字需要转化成相关的记录编号时，发生冲突的可能性是常常存在的．散列技术就是设计用来减少发生冲突的机会．但是，一旦发生了冲突，程序必须能够将冲突记录地址转化成磁盘上的某些唯一空间中．

§13.6　二进制文件：简述

二进制文件按照存储字节顺序建立的，而没有考虑 ASCII 字符、数字或者串、记录长度、回车等等．要建立一个二进制文件，用户可简单地指出哪些字节要写，在文件中它们走到哪里．从二进制文件中读时，用户需要有相同的规范．

不同版本上的 Basic 或者其它应用程序包需要访问文件时，此文件通常要建立二进制文件．

§ 13.7 本章辅导

§ 13.7.1 本章小结

Ⅰ 主文件处理

A. 主文件是某专用应用程序的主要数据集.
   1. 主文件可以是随机的，也可是顺序的.
   2. 主文件通常存贮在磁盘上.

B. 不管文件是顺序的，还是随机的，许多主文件过程是相同的.
   1. 最初，主文件是由有效数据建立的，以保证数据错误最少.
   2. 建立更变记录的细目文件.
   3. 通过细目文件修改主文件，细目文件中包括所有对主文件进行的更变.
   4. 从主文件中可产生报表.

C. 可购买到数据库管理系统（DBMS）中，包含有关于某些标准文件操作. 但是专用操作需要客户用语言如 Turbo Basic 来编写程序.

Ⅱ 顺序文件的修改

A. 顺序文件必须按顺序处理

B. 顺序主文件和细目文件的记录长度是可变化的，记录中的字段则由逗号分隔开.
   1. 打开文件用作INPUT或者OUTPUT.
   2. 用WRITE#（文件号）在文件上建立记录.
   3. 用INPUT#（文件号）从文件中读记录.
   4. 作业结束时关闭文件.

C. 修改过程保持主文件最最新的.
   1. 应打印出记录跟踪以检验所作的更变是正确的.
   2. 这一过程需要旧主文件和细目各文件作为输入，而产生新主文件作为输出.
   3. 旧主文件、细目文件和新主文件中的记录都必须按同一关键字段顺序排列.
   4. 必须编写关于细目关键字值小于、大于和等于旧主文件关键字值的处理过程.
   5. 必须处理细目文件和主文件直到两个文件都结束为止.

Ⅲ 修改过程中的有效性检验

A. 不论何时，向文件增加新记录，在细目记录中都应用一编码字段，以保证这是一笔新账目。

B. 通过比较记录中的其它字段，可以双检验关键字段（如:如果SSNo 相同,则检查 Employe Name 以核实)

C. 不论何时，从文件中删除记录，在细目记录中应用一编码字段来检验.

Ⅳ 关于每条主记录有多个细目记录的顺序修改过程

A. 对单个主记录，将可能有多个细目记录.

B. 直到关于主记录的所有细目记录处理完后，才能填写新的主记录.

C. 为了检验，可以用DOS命令TYPE来显示数据文件内容.

Ⅴ 随机文件构造

A. 构成随机文件，以便可以直接检索记录，而不需顺序访问.

B. 用关键字段来确定记录位置。

1. 某些字段的值可能可直接用作关键字段.
2. 通过应用某一算法可将某一字段的值转换成相关的记录编号.

C. 顺序文件与随机文件的比较

1. 顺序文件用的是可变长度记录，由用逗号分隔的字段来给记录分界；而随机文件用的是固定长度记录.
2. 当有相同的记录长度时，因为不需逗号，所以随机文件占用的空间比顺序文件少.
3. 随机文件不能象顺序文件那样可移植.
4. 在随机文件中不用引号来给字符串变量定界.
5. 如果需要直接访问记录，或者如果在文件中只有小数目的记录需要一次处理时,建立随机文件是很合适的.

D. 随机文件处理

1. OPEN语句中必须指明记录长度.
2. OPEN语句不指明用作输入还是输出，因为关于输入和输出，随机文件均可用在同一计算机程序中运行.
3. 每个字段必须用FIELD语句定义.
4. 在随机存取文件中的所有字段必须用RSET和LSET来对齐.
5. 所有的数值字段均必须用"make"函数将其转换成字符串字段.
6. 用PUT语句将记录写进随机文件中.
7. 在作业结束时，必须关闭随机文件.
8. 用GET语句即可顺序地存取记录又可随机存取记录.
9. 应在错误控制规程中处理输入／输出错误.
10. 可用散列算法将关键字值转换成相对应的记录编号.

§ 13.7.2 本章自测

1. 顺序文件和随机存取文件之间的主要区别是什么?
2. 主文件建立程序的主要目的是: 确保________.
3. 在________方式下，通常处理顺序文件；而在________方式下，通常处理随机存取文件.
4. 修改一个顺序主文件需要________文件、________文件和________文件.
5. 在顺序修理过程中，细目记录的关键字段和________文件中关键字段进行比较.
6. 当随机文件中的关键字段需转换成相对应的记录编号时，要使用________.
7. 说明MKI$、MKS$和MKD$函数之间的区别.
8. 在GET语句中，如果它没有包括记录编号，则会发生什么情况?
9. 写出FIELD语句，确定一个3位数字用作顾客编号（整型)、20个字符用作顾客姓名、20个字符用作住址和6位数字用作结余的区域.
10. 编写一段程序，用来转化这些数据并写进题9定义的随机文件中. 假设相对应的字段已经输入.

§ 13.7.3 实例

考虑下列两个程序，图 13.9 建立一个随机库存文件；图 13.10 利用菜单查询有关记录，修改记录和产生来自随机文件的报表; 图 13.11 说明了修改程序的层次图.

```
'*************************************************************
'This program creates a randomly accessible file
'        for use in an inventory system
'*************************************************************
CLS
PRINT "Inventory file creation program"
KILL "INVENTRY.BAK"                      'Remove old backup
NAME "INVENTRY.DAT" AS "INVENTRY.BAK"    'Backup old file
OPEN "INVENTRY.DAT" AS #1 LEN = 34       'Create new file
FIELD #1, 20 as Desc$, 2 as Size$, 6 as ItemColor$, 4 as Price$, 2 as Qty$
FOR I% = 1 to 15
        PRINT "Record: ";I%
        PRINT "Enter Desc, Size, Color, Price, Qty"
        INPUT DescIn$, SizeIn%, ItemColorIn$, PriceIn, QtyIn%
        LSET Desc$ = DescIn$
        LSET Size$ = MKI$(SizeIn%)               'Convert to string
        LSET ItemColor$ = ItemColorIn$
        LSET Price$ = MKS$(PriceIn)              'Convert to string
        LSET Qty$ = MKI$(QtyIn%)                 'Convert to string
        PUT #1, I%                               'Write the record
NEXT I%
'End of program
PRINT "Done."
CLOSE #1
END
```

图 13.9(a) 建立随机库存文件程序

Pants	33	Blue	11.95	3
Pants	34	Grey	10.95	3
Pants	34	Black	10.95	1
Jeans	34	Blue	7.00	2
Jeans	36	Blue	7.00	3
Jeans	36	Grey	7.50	2
Shirt	14	White	5.00	2
Shirt	14	Blue	5.00	3
Shirt	15	Blue	5.50	1
Shirt	16	White	5.00	2
Sweater	15	Yellow	10.00	1
Sweater	16	Green	12.50	1
Sweater	16	Grey	12.00	1
Jacket	42	White	15.00	2
Jacket	42	White	13.00	1

图 13.9(b)　建立的随机库存文件

```
'************************************************************************
' This program updates an inventory file
'************************************************************************
OPEN "B:INVENTRY.DAT" AS #1 LEN = 34     'Open existing file
FIELD #1, 20 as Desc$, 2 as Size$, 6 as
          ItemColor$, 4 as Price$, 2 as Qty$
Choice% = 0                                        'Initialize before loop

DO WHILE Choice%
        Call Menu                                  Display the choices
        INPUT Choice%
        SELECT CASE Choice%
        CASE 1
                CALL Inquiry(Desc$, Size%, ItemColor$, Price, Qty%)
        CASE 2
                CALL Report(Desc$, Size%, ItemColor$, Price, Qty%)
        CASE 3
                CALL WrapUp
        CASE ELSE
                PRINT
                PRINT "Illegal choice..."
                INPUT "Press Enter to continue ";Pause$
        END SELECT
LOOP
END
'*****************************************************************
' This procedure prompts the user for input and determines
' if a record us to be modified or updated
'*****************************************************************
SUB Inquiry(Desc$, Size%, ItemColor$, Price, Qty%)
        LOCAL N, Choice$
        INPUT "Item No. ";Rec%
        CALL GetRecord(Rec%, Desc$, Size%, ItemColor$, Price, Qty%)
        PRINT
```


R158-32②

```
        CALL Heading
        CALL DisplayLine(Rec%, Desc$, Size%, ItemColor$, Price, Qty%)
        PRINT
        INPUT "Modify this record (Y/N) ";Choice$
        If Choice$ = "Y" or Choice$ = "y" THEN
                CALL Modify(Rec%, Desc$, Size%, ItemColor$, Price, Qty%)
        END IF
END SUB
'**************************************************************
' This procedure modifies or updates a record
'**************************************************************
SUB Modify(Rec%, Desc$, Size%, ItemColor$, Price, Qty%)
        SHARED Size$, Price$, Qty$                    'Un passed fields are global
        LOCAL Choice$
        INPUT "Description: ";DescIn$
        INPUT "Size       : ";SizeIn%
        INPUT "Color      : ";ItemColorIn$
        INPUT "Price      : ";PriceIn
        INPUT "Quantity   : ";QtyIn%
        LSET Desc$ = DescIn$
        LSET Size$ = MKI$(SizeIn%)                    'Convert to string
        LSET ItemColor$ = ItemColorIn$
        LSET Price$ = MKS$(PriceIn)                   'Convert to string
        LSET Qty$ = MKI$(QtyIn%)                      'Convert to string
        PUT #1, Rec%                                  'Write the record
END SUB
'**************************************************************
' This procedure prints output
'**************************************************************
SUB Report(Desc$, Size%, ItemColor$, Price, Qty%)
        LOCAL Pause$, Total
        CLS
        PRINT "                          Inventory Report "
        PRINT
        CALL Heading
        FOR I% = 1 to LOF(1)/34
                CALL GetRecord(I%, Desc$, Size%, ItemColor$, Price, Qty%)
                CALL DisplayLine(I%, Desc$, Size%, ItemColor$, Price, Qty%)
                Total = Total + Price * Qty%
        NEXT I%
        PRINT
        PRINT USING "      Total value = $$$$$#.##";Total
        PRINT
        INPUT "Press Enter to continue ";Pause$    'permits user to pause
END SUB

'**************************************************************
' This procedure accesses a record from the file
'**************************************************************
SUB GetRecord(Rec%, Desc$, Size%, ItemColor$, Price, Qty%)
        SHARED Size$, Price$, Qty$                    'Un-passed fields are global
        GET #1, Rec%
        Size% = CVI(Size$)                            'Convert to numeric
        Price = CVS(Price$)
        Qty% = CVI(Qty$)
END SUB
'**************************************************************
' This procedure prints headings
'**************************************************************
SUB Heading
        PRINT "     Item  Description          Size   Color    Price   Qty"
        PRINT "     ---------------------------------------------------------"
END SUB
'**************************************************************
' This procedure displays a line of output
'**************************************************************
SUB DisplayLine(Rec%, Desc$, Size%, ItemColor$, Price, Qty%)
     PRINT USING "    ###    \                    \ ####   \      \  $$#.##  ###";
                      Rec%;Desc$;Size%;ItemColor$;Price;Qty%
END SUB
'**************************************************************
' This procedure performs end-of-job functions
'**************************************************************
SUB WrapUp
        CLOSE #1
        PRINT
        PRINT "Done."
END SUB
```

```
'********************************************************************
' This procedure prints a menu
'********************************************************************
SUB Menu
        CLS
        PRINT "                    Inventory Management System"
        PRINT
        PRINT "          1) Inquiry"
        PRINT "          2) Report"
        PRINT "          3) Exit"
        PRINT
        PRINT "Enter your choice ===> ";
END SUB
```

图 13.10(a)　修改库存文件程序

Inventory 文件(一)

```
                    Inventory Report

Item  Description              Size   Color    Price   Qty
-----------------------------------------------------------
  1   Pants                     33    Blue    $11.95    3
  2   Pants                     34    Green   $10.95    3
  3   Pants                     34    Black   $10.95    1
  4   Jeans                     34    Blue     $7.00    2
  5   Jeans                     36    Blue     $7.00    3
  6   Jeans                     36    Grey     $7.50    2
  7   Shirt                     14    White    $5.00    2
  8   Shirt                     14    Blue     $5.00    3
  9   Shirt                     15    Blue     $5.50    1
 10   Shirt                     16    White    $5.00    2
 11   Sweater                   15    Yellow  $10.00    1
 12   Sweater                   16    Green   $12.50    1
 13   Sweater                   16    Grey    $12.00    1
 14   Jacket                    42    Black   $15.00    2
 15   Jacket                    42    Blue    $13.00    1

Total value =   $247.65

Press Enter to continue
```

图 13.10(b)　修改前的库存文件

记录调整运行

```
                  Inventory Management System

          1) Inquiry
          2) Report
          3) Exit

Enter your choice ===> ? 1
Item No. ? 12

     Item  Description            Size   Color    Price    Qty
     ---------------------------------------------------------
     12    Sweater                 16    Green   $12.50     1

Modify this record (Y/N) ? Y
Description: ? Sweater
Size       : ? 16
Color      : ? Green
Price      : ? 15.00
Quantity   : ? 5
```

```
                    Inventory Management System

         1) Inquiry
         2) Report
         3) Exit

Enter your choice ===> ? 1
Item No. ? 1

     Item  Description          Size  Color    Price    Qty
     ------------------------------------------------------
      1    Pants                 33   Blue     $11.95    3

Modify this record (Y/N) ? Y
Description: ? Pants
Size        : ? 33
Color       : ? Blue
Price       : ? 12.50
Quantity    : ? 25
```

```
                    Inventory Management System

         1) Inquiry
         2) Report
         3) Exit

Enter your choice ===> ? 1
Item No. ? 3

     Item  Description          Size  Color    Price    Qty
     ------------------------------------------------------
      3    Pants                 34   Black    $10.95    1

Modify this record (Y/N) ? N
```

```
                    Inventory Management System

         1) Inquiry
         2) Report

         3) Exit

Enter your choice ===> ? 2
```

图 13.10(c)　程序运行过程

Inventory 文件(二)

```
                    Inventory Report

Item  Description          Size  Color    Price    Qty
------------------------------------------------------
 1    Pants                 33   Blue     $12.50    25
 2    Pants                 34   Green    $10.95     3
 3    Pants                 34   Black    $10.95     1
 4    Jeans                 34   Blue      $7.00     2
```

```
 5     Jeans                  36    Blue      $7.00    3
 6     Jeans                  36    Grey      $7.50    2
 7     Shirt                  14    White     $5.00    2
 8     Shirt                  14    Blue      $5.00    3
 9     Shirt                  15    Blue      $5.50    1
10     Shirt                  16    White     $5.00    2
11     Sweater                15    Yellow   $10.00    1
12     Sweater                16    Green    $15.00    5
13     Sweater                16    Grey     $12.00    1
14     Jacket                 42    Black    $15.00    2
15     Jacket                 42    Blue     $13.00    1

Total value =    $586.80

Press Enter to continue ?
```

图 13.10(d)　修改后的库存文件

最终的屏幕显示

```
                Inventory Management System

     1) Inquiry
     2) Report
     3) Exit

Enter your choice ===> ? 3

Done.
```

图 13.10(e)　最终结果

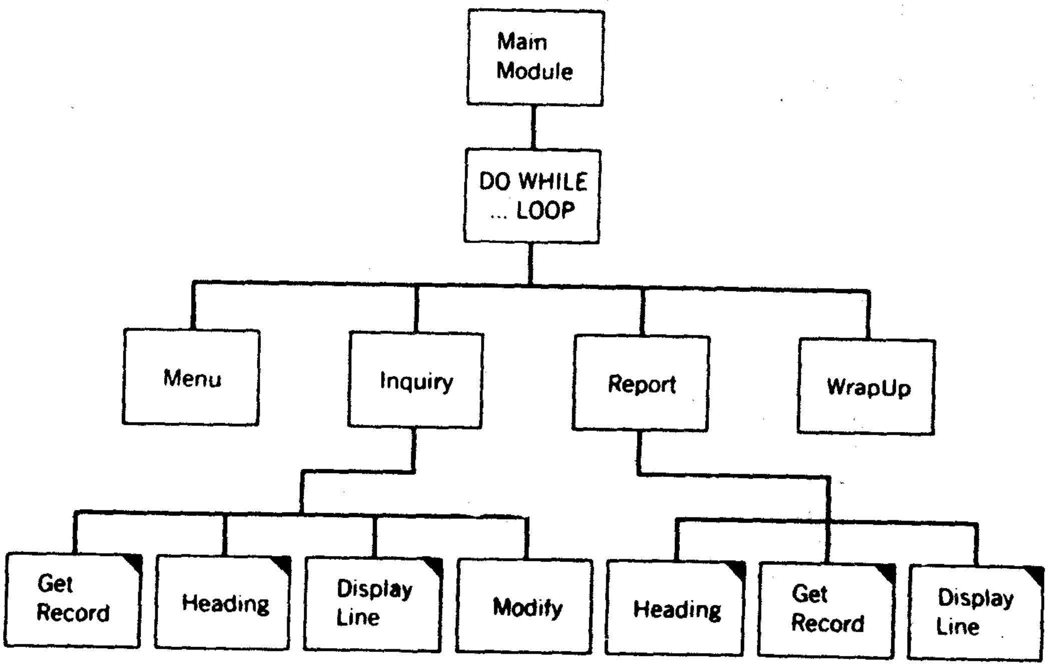


图 13.11　修改程序的层次图

§ 13.7.4 复习题

Ⅰ. 一般问题

1. 对一个随机文件，用OPEN和FIELD语句编码，包含有下列字段.

Student Id(学号)	整型	4
Student Name(学生姓名)	字符串	20
Social Security number(社会保险号)	字符串	9
Units(单元)	整型	2
GPA	单精度	3

2. 文件中活动记录数每天在0～50之间变化，每条记录包含有EmployeeId（3个字节数字）、Emplyee name （20个字符）、Hours worked （3个字节数字） 和 Pay Rate （4位数）首先，输入一个值（NoRec%），指明了当天的活动记录编号，写一段程序读 NoRec%. 然后动态地建立一张表存放雇员姓名和总支付额. 顺序地读每条记录，计算总支付额，在表中放置合适的变量.
3. 说明RSET 和LSET 函数之间的区别.

Ⅱ 数据确认

修改实例，使它们包括合适的编码，用来:①测试所有的错误;②打印全体控制列表(处理的记录、遇到的错误、分批总数).

§ 13.7.5 调试练习

1. 考虑如下程序:

```
CLS
OPEN "Payroll.Dat" AS #1
FIELD #1, 2 AS EmployeeId$, 20 AS EmployeeName$,
        4 AS Salary
PRINT "Enter employee id (-99 when done)"
INPUT EmployeeId$
DO WHILE EmployeeId$ <> -99
    GET #1
    PRINT "Employee Name "; EmployeeName$
    PRINT "Salary "; Salary$
    PRINT "Enter employee id (-99 when done)"
    INPUT EmployeeId$
LOOP
END
```

a. 更正OPEN语句.

b. 更正GET语句.

c. 更正所有其它错误.

更正下列程序:

```
2. OPEN "AcctsPay.Dat" AS #1 LEN = 30
   FIELD #1, AcctNo$ AS 4, AcctName$ AS 20, BalDue$ AS 4
3. INPUT BalDue
   LSET BalDue$ = MKI$(BalDue)
4. Input AcctNo$, AcctName$, BalDue$
   PUT #1, AcctNo$, AcctName$, BalDue$
5. INPUT AcctNo%
   GET #1, AcctNo%
   AcctNoX = CVI(AcctNo$) THEN
   IF  AcctNo% = AcctNo
       PRINT "Balance Due is "; BalDue$
   END IF
```

§ 13.7.6　编程题

1. 编写程序建立随机文件包含有学生证号（4个整型数字，作用关键字段）、学生姓名（20个字符）和GPA（3个数字）用户采用自己的样本数据，最少建立有30条记录，包括有活动状态字段.
2. 编写程序，从题1中的文件里顺序地读所有的活动记录，且计算出所有GPA大于3.0的学生数.
3. 编写一程序，从题1建立的文件中随机地存取记录把学生证号当作关键字段输入且打印出学生姓名和GPA.
4. 编写程序，利用题1建立的文件，随机地修改学生的GPA. 修改过程应该既能够添加记录或者删除记录，又能够更改记录.
5. 编写程序，用来建立和处理包含有姓名，电话号码和地址的随机文件，程序对输入任意姓应可显示出地址和电话号码. 记录关键字段是采用散列算法从电话号码中派生而来的.
6. 编写程序，用来建立和处理一个关于小酒巴（15个房间)的顺序文件，其中包含有房间号，日住房率和住宿姓名，然后编写一修改程序，能更变居住者的房号、显示住房率和任意一个房间的居住者.
7. 考虑下列顺序主文件.

   MparNo, QtyOnHand

   考虑下列细目文件:

   TportNo, TQty, Tcode

TCode = 1　　删除记录

Tcode = 2　　添加记录

　　对于要添加的记录而言，TQty 应等于 QTyOnHand。

Tcode = 3　　修改记录

　　因为 TQty 等于计划的修改量，所以 TQty 应从 QtyOnHand 中提取。

顺序修改主文件，假设对每对记录有多项细目记录，显示主文件中的任意零件数，现有的零件数量应少于 50。

第十四章　图形和声音

本章要求:

1. 学会产生图形输出.
2. 熟悉Turbo Basic中产生图形输出的指令
3. 学会用Turbo Basic产生声音和音乐.

§14.1　Turbo Basic 图形概述

本书重点在于用 Turbo Basic 产生打印报表和显示数据形式的输出. 这种类型的输出通常由字符串和数值组成, 供行政人员、管理人员和操作人员使用.

对于许多应用程序而言, 以图形形式描述的结果或输出可能更有意义和更有用. 那些行政人员和其他用户为作出决定而必须分析结果的应用程序尤其如此. 对于行政管理人员来说,"一幅图形"的确"顶一千句话". 有许许多多的软件包可为商业和其他方面应用程序将数字数据转换成高质量的图形.

Turbo Basic 程序设计语言含有大量的、能产生高质量图形的指令. 不过, 请记住: 用户必须拥有相应的硬件, 以显示或/和打印彩色或单色 (如黑白) 图形. 此外, 还应注意, 数字数据到图形的转换要求算法, 几何学及其他数学关系的知识. 一旦掌握了产生图形的指令, 弄清楚了有关要素, 用户便可使用 Turbo Basic 画图和产生任意的图形字符和动画. 它们可用于仿真、建模以及游戏等方面.

要提供 Turbo Basic 如何将数字数据转换成图形的详尽解释, 需要一本这方面的专用书籍. 的确, 有许许多多专门或重点介绍图形的 Basic 书籍. 本章仅概述可用于图形输出的 Turbo Basic 指令.

我们将重点放在下述各类图形.

各类图形		
类型	**图例**	**典型用处**
直方图	图 14.1	显示数量间的关系. 例如, 可显示不同部门间的费用变化情况.
馅饼图	图 14.2	显示整体的划分情况. 例如, 可显示公司内部各部门的创收情况.
线条图	图 14.3	显示一个或多个数量的变化趋势. 例如, 可显示一个季度内公司的发展情况.

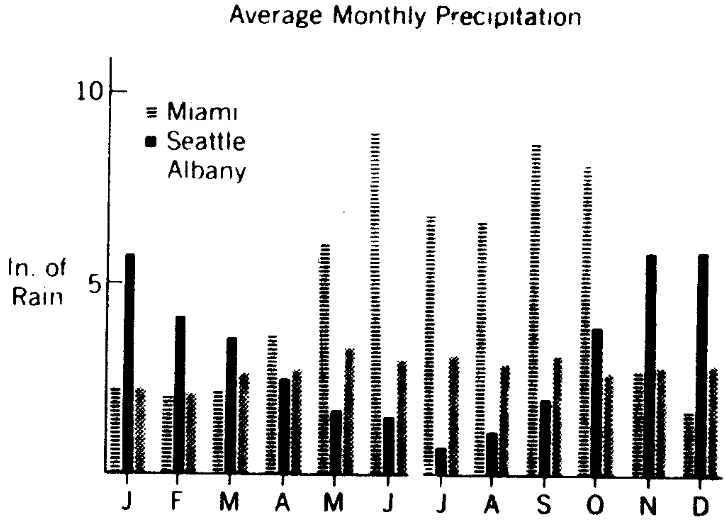


图 14.1 直方图

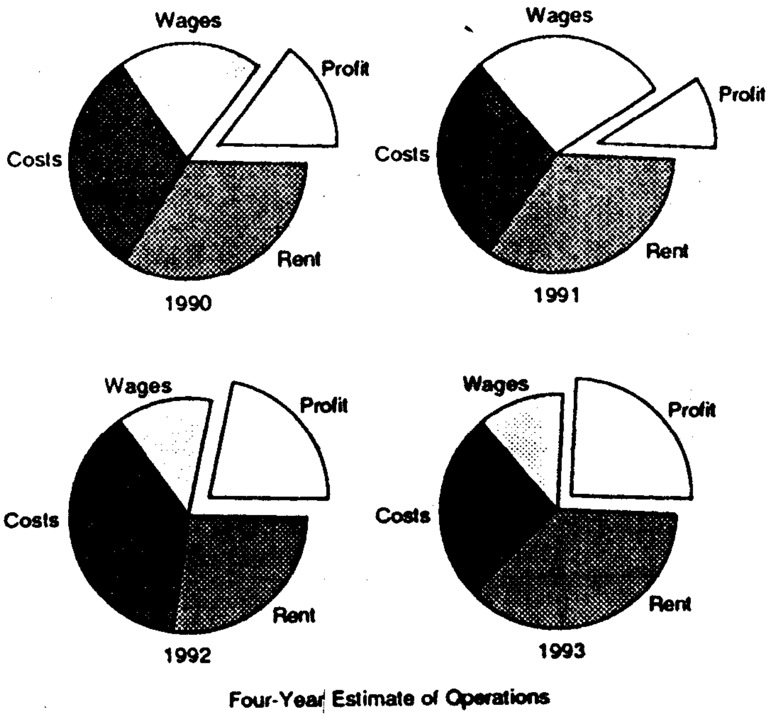


图 14.2 馅饼图

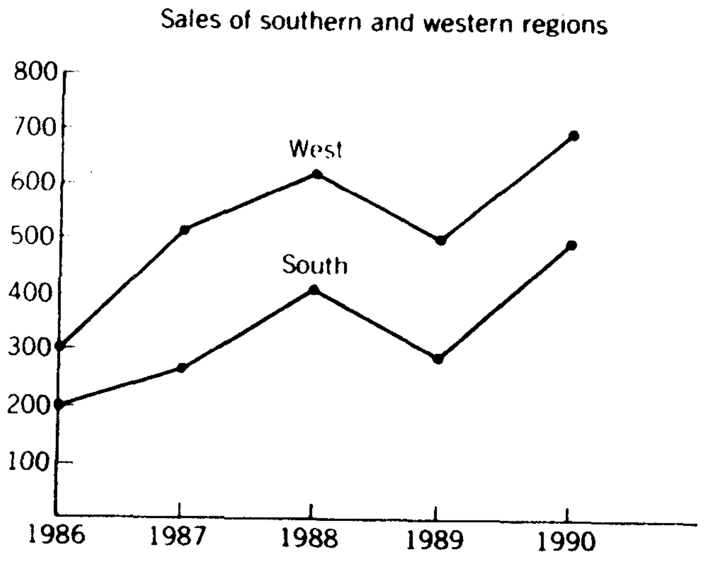


图 14.3 线条图

在讨论产生这些图形所需指令之前，先着眼于硬件需求.

§ 14.2 显示图形所需的微型设备

§ 14.2.1 定义

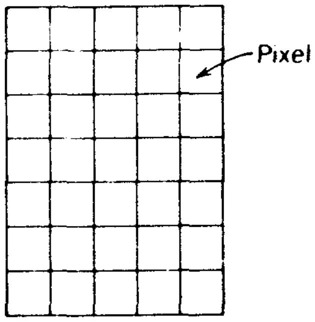
为了显示图形字符，用户必须拥有一台图形监视器和一个图形适配器．本书交替使用监视器，屏幕和 CRT（Cathode Ray Tube —阴极射线管）等术语．虽然图形监视器和适配器是许多 IBM 兼容机和 IBM Personal System／2 的标准设备，但它们不是 IBM PC 机的标准设备．为了显示彩色图形，还要一个彩色图形监视器（graphics monitor）（注：Turbo Basic 1.0 版不支持 Hercules 图形，但 Turbo Basic 后期版本支持了）．因而，用户可能只具备单色 CRT 的图形能力，但为了真正获得最多的图形显示功能，应当拥有一台彩色 CRT．

使用某些计算机时，为了使用图形指令，用户必须先执行一条 DOS 命令．一般是在 DOS 状态下执行 GRAPHICS 命令，切换到图形模式．这样方可显示图形字符和正文（字母和数字）．

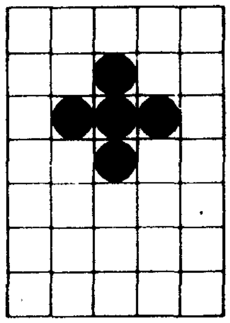
§ 14.2.2 监视器的类型

虽然有些监视器在图形模式下每行只能显示 40 个字符，但大多数监视器每行显示 80 个正文字符，每屏显示 25 行．本章重点介绍 80×25 的监视器．另一个度量监视器的方法是象素（pixels）或图形元素的数目．象素是屏幕上一个用来构造各个字符或图形符号的

独立点．图 14.4 是象素的放大图．屏幕上可编址或访问的象素越多，字符的清晰度或分辨率（Resolution）越高．中分辨率屏幕常常可为图形从 320×200 至 640×200（或更多）个象素编址；高分辨率屏幕通常可为图形从 640×200 至 720×480（或更多）个象素编址．



(a) 屏幕上单个位置的放大图(5×7 点阵)



(b) 用象素构造用户自己的图形字符

图 14.4 象素

单色屏幕提供最高的清晰度或分辨率，显示要求可读性很好的正文时很理想．但是，单色屏幕显示图形字符的能力受到限制．另外，它还不能显示可使图形非常有趣的颜色．

Turbo Basic 支持微型机及其兼容机上的五种主要级别的视频接口：Monochrome Display Adapter（单色显示适配器），它只能显示单色正文；Color Graphics Adapter（CGA—彩色图形适配器）、Enhanced Graphics Adapter （EGa—增强型图形适配器）Multi-Color Graphics Array Adapter （MCGA—多色图形陈列适配器）；以及 Video Graphics Array （VGA—视频图形陈列），它适用于 PS／2，50，60，70，80 型．后四种可处理彩色、黑白正文和图形．

表 14.1 给出了适用于 IBM PC 和 IBM Personal System／2 的各种图形适配器的详细情况．

Table 14.1

Comparison of Graphics Capability

	Monochrome/ Printer Adapter (used by PCs)	Color Graphics Adapter (used by PCs)	Enhanced Graphics Adapter (EGA) (used by PCs)	Multi-Color Graphics Array Adapter (MCGA)	Video Graphics Array (VGA) (used by PS/2 Models 50/60/70/80)
Graphics Memory Size	4 K	16 K	256K*	256K	256K
Best Alphanumeric Modes					
Maximum resolution	720 × 350	640 × 200	640 × 350	640 × 480	720 × 400
Maximum colors	2 (black and white)	16	16	16	16

除非另外说明，否则最大分辨率和最多颜色数被同时支持.

* EGA的标准图形内存体积为64K；通过图形内存扩展选择，它可扩展到256K，支持16种颜色.

§ 14.3 Turbo Basic 图形指令

§ 14.3.1 Screen 指令

取决于所用的监视器，屏幕可设置为：在中分辨率或高分辨率模式下，显示图形字符和／或正文. 下面的 SCREEN 指令将设定屏幕显示模式，以便于屏幕上图形的显示.

模式	效 果
0	不改变宽度，设定文本模式。
1	设定中分辨率图形模式，将屏幕宽度改变为 40 个字符.

2	设定高分辨率图形模式，将屏幕宽度改变为 80 个字符.
7	在 EGA 上设定中分辨率模式，允许使用 16 种颜色和 640×200 个象素.
8	在 EGA 上设定高发辨率模式，允许使用 16 种颜色和 640×200 个象素.
9	在 EGA 上设置增强型高分辨率模式，可以有 64 种调色板，每种允许 4 或 16 种颜色（取决于 EGA 的存贮内存）和 640×350 个象素.
10	在带有单色监视器的 EGA 上设定高分辨率模式，允许 4 种属性和 640×350 个象素.
11	在带有单色监视器的 VGA 上设定高分辨率模式，允许 4 种属性和 640×480 个象素.
12	在带有彩色监视器的 VGA 上设定高分辨率模式，允许 16 种颜色和 640×480 个象素.

用户将使用的 SCREEN 指令取决于所拥有的图形适配器，参见表 14.2.

表 14.2　屏幕模式样例

适配器	屏幕模式	正文格式	说　明
单显	0	80 列	80×25 个字符，具有四种属性（正常，高亮度，闪烁，下划线）.
CGA	0,1,2	40／80 列	屏幕分辨率为：640×200 个象素. 在正文模式下最多可用 16 种颜色；在图形模式下最多可用 4 种颜色.
EGA	0,1,2, 7,8,10	40／80 列	屏幕分辨率可达 640×350 个象素. 有 64 种调色板，每种最多有 16 种颜色（取决于 EGA 显示内存）.
MCGA	0,1,2,11	40／80 列	MCGA 的屏幕分辨率为 640×480 个象素；屏幕模式 11 为黑白模式.
VGA	0,1,2,12	40／80 列	屏幕分辨率可达 640×480 个象素，最多有 16 种颜色.

本章给出的演示图形均使用“SCREEN 2”. 在给定的程序中，用户若想回到正文模式，可用 SCREEN.

还需注意，SCREEN 语句将清屏。因此，使用 SCREEN 语句时，没有必要使用 CLS 指令。

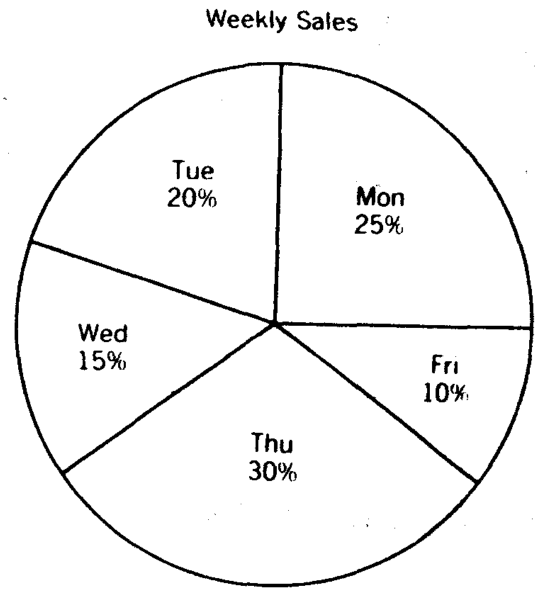
§ 14.3.2 馅饼图

§ 14.3.2.1 图形设计

描绘用总面积的 100%表示的总体或"馅饼"的分割情况时，馅饼图是一种非常有用的方法。假如要打印下述某周的日销售量：

日期	总的销售量
MON	250
TUE	200
WED	150
THU	300
FRI	100

经理可能觉得以馅饼图形式显示的数据更有用：



这个馅饼图给出了该周内每天成交的销售量占总量的百分比。

用 Turbo Basic 画如上所述的馅饼图是很容易的。首先，确定图形在屏幕上的位置及其大小。这些因素取决于所使用的屏幕模式是中分辨率的，还是高分辨率的。如果所用的模式为高分辨模式，那么如图 14.5 所示设计馅饼图。这个图形的设计使用了类似于打印

机空格表（Printer Spacing Chart）的 CRT 格式纸（Lagout Sheet）。

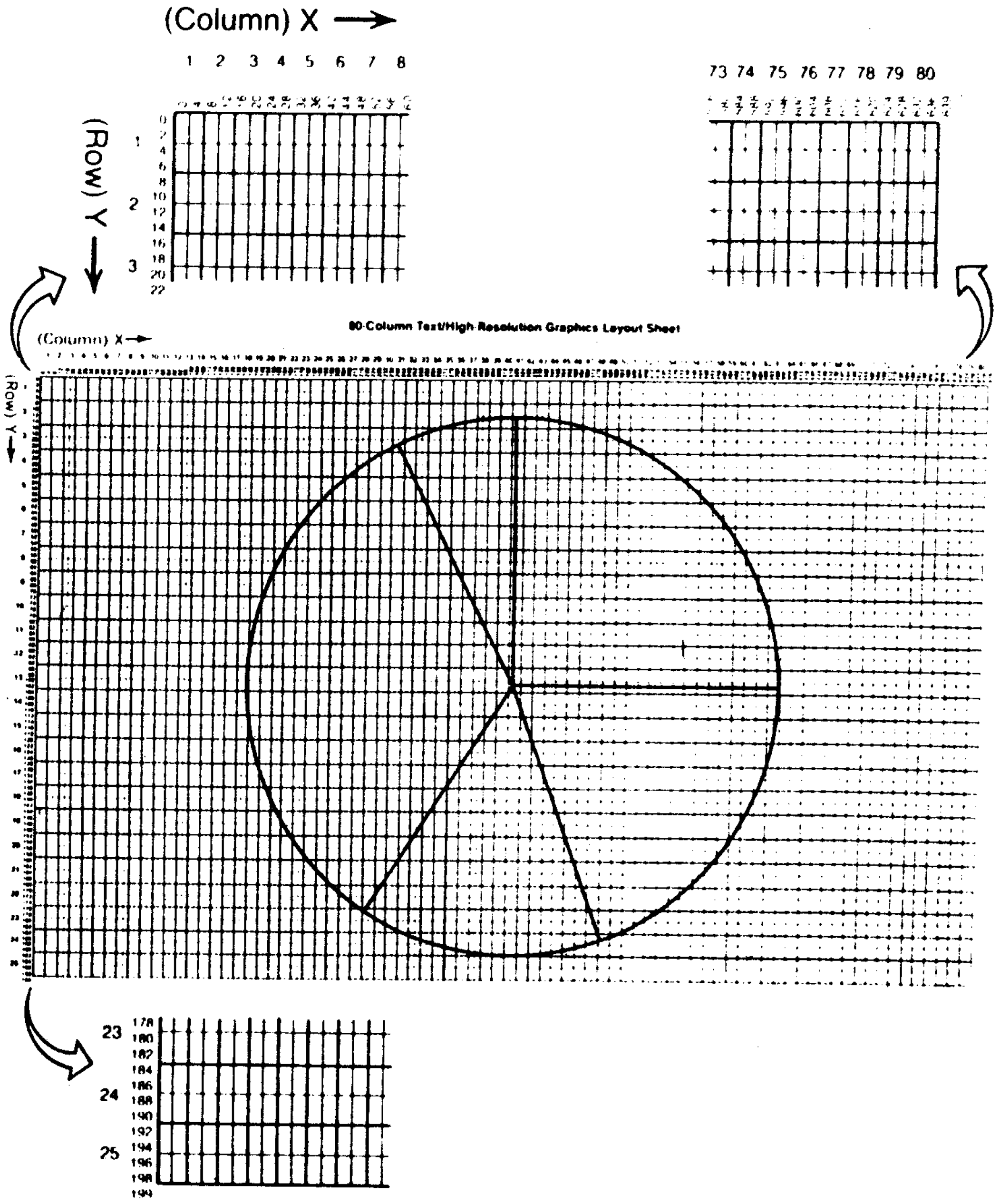


图 14.5 用 CRT 格式纸设计馅饼图

注意：本图中 CRT 格式表分成 640×200 个点或象素。纵向象素标号为 0–639，横向象素的标号为 0–199。

虽然其他类型的屏幕可编址的象素或多或少，但这个 CRT 格式屏可用于某些监视器。如前所述，象素是屏幕上一个用来创建图形符号的独立点。屏幕上可编址或访问的象素越多，字符的清晰度或分辨率就越高。我们已经知道，高分辨率屏幕可编址的象素比中分辨率屏幕的多。本书给出的图形例子均假定使用带 640×350 个可编址象素的高分辨率

屏幕.

§14.3.2.2 CIRCLE 指令

CIRCLE 语句指定:

* 1. 园心.
* 2. 半径 (它确定了园的面积)
  3. 园的颜色 (可使用的颜色取决于所用监视器的类型); 它是任选项.
* 4. 起始弧度和终止弧度
  5. 园的园度. 如果指定的监视器要产生一个不太园的图形显示, 如椭园, 园度才有用. 在此从略.

因为画馅饼图只需要加"*"号的元素 (1、2和4), 所以只重点介绍它们.

若使用高分辨率屏幕, 馅饼图的中心定于象素 (32, 100).

为了画馅饼图的扇形, 应编程计算它的相应角度. 根据给出的每周销售量, 星期一的销售额为 250 美元, 总销售额为 1000 美元. 因此, 星期一的销售额占总销售额的 25% (即 360° 的 1/4 等 90°). 类似地, 星期一的销售额是本周总销售额 1000 美元中的 200 美元, 因而它对应的扇形角度应为 72° (因为总数的角度为 360°, 它的 20%是 72°).

若各扇形的角度从 0 (即园心) 算起, 则各扇形的起始点为 0, 然后, 按下式计算扇形角度;

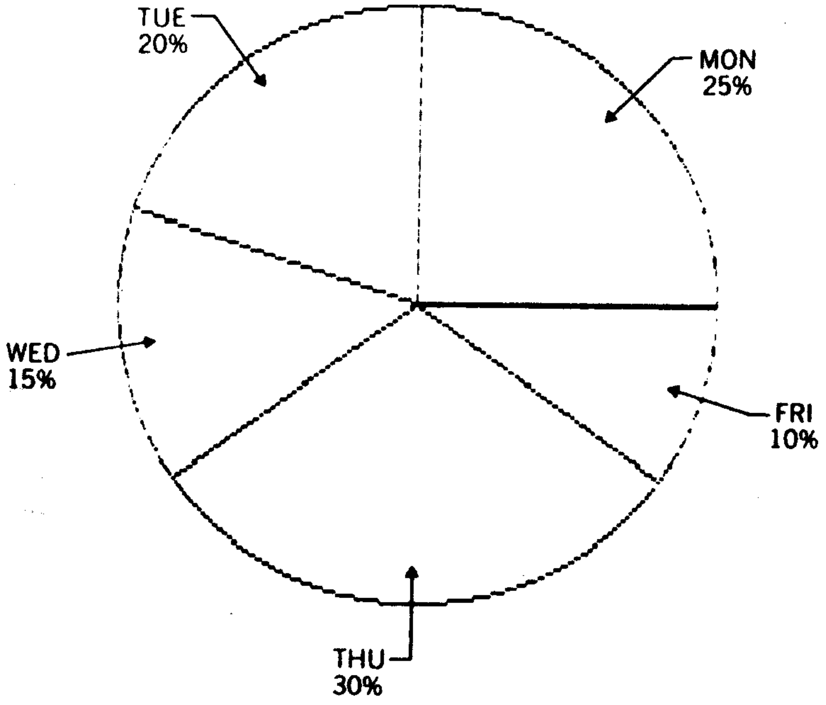
0− (各扇形对应的值/所有值的总和) * 2 * Pi

不管要描述的数值个数和总数如何, 这个公式适用于任何馅饼图. 另一个方法是, 将一个扇形的起始点定于比 0 稍小的负数: −0.000001. 这样可将扇形放在 X 轴的正数一边.

绘制上述销售情况馅饼图的程序如下:

```
SCREEN 2
COLOR1 = 1             Sets the color (monochrome)
X = 320                Sets pixel
Y = 100                Sets pixel
Pi = 3.14159           Sets pi
A2 = -.000001          Sets starting point for each wedge
Radius = 180           Sets radius
TotalSales = 0
OPTION BASE 1
DIM DailySales (5)
 ************************************************************
         Reads in 5 daily sales figures using READ & DATA
 ************************************************************
FOR I% = 1 TO 5
         READ DailySales (I%)
         LET   TotalSales  = TotalSales + DailySales (I%)
NEXT I%
 ************************************************************
         Draws wedges for each daily sales figure
 ************************************************************
FOR I% = 1 TO 5
         A1 = A2
         A2 = A1 - DailySales (I%) / TotalSales * 2 * Pi
         CIRCLE (X,Y), Radius,Color1,A1,A2
NEXT I%
DATA 250,200,150,300,100
END
```

该程序产生如下所述的馅饼图:



该程序先用 SCREEN 2 清屏，并将屏幕设定为图形模式，然后，将颜色设为 1 或单色。如果用户具有彩色监视器，可以使用其他颜色。通过将 X 设为 320、Y 设为 100，园心定于（320，100）。CIRCLE（X，Y）将园心确定在（320，100）处的象素上；Pi 取为 3.14159；各扇形的起始点稍微偏离中心，定于−0.000001 处，这是因为象素是一个并不精确地位于园心的点；半径设为 180；接着，将 Totalsales 和数组初始化，并随即用 READ 和 DATA 语句读入销售金额；最后，FOR…NEXT 循环逐个画出五个每日对应扇形。

下节将介绍如何给各元素起标号。用上述结构作指导，用户可绘制任意的馅饼图。

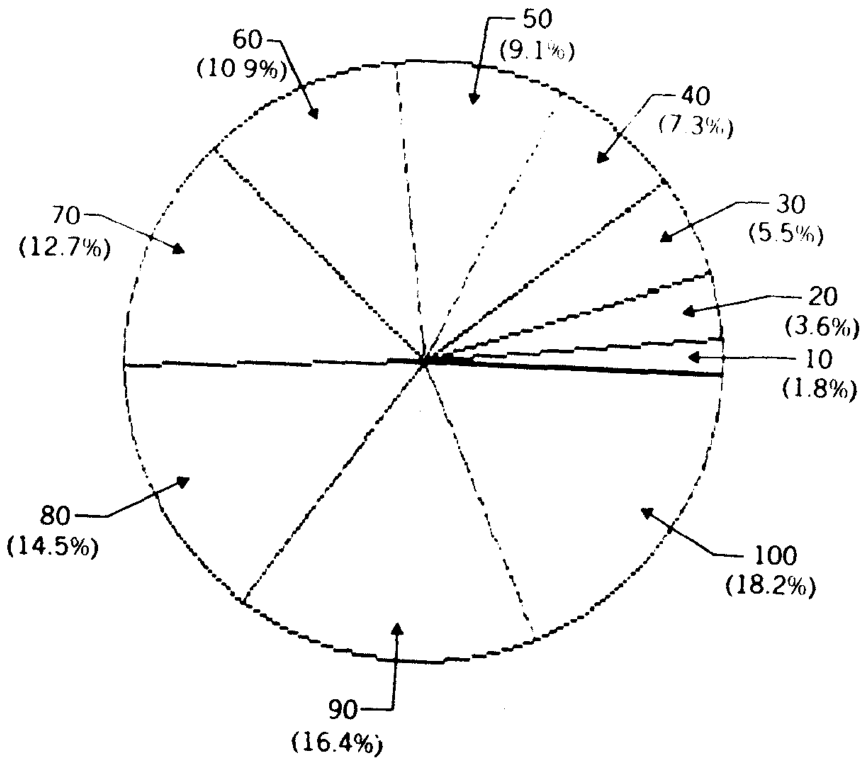
假设用户要用 10 个数值画馅饼图:

10，20，30，40，50，60，70，80，90，100

十个数值的总和为 555。用 READ 和 DATE 语句读入这十个数值，然后用上述的 CIRCLE 指令显示图形:

```
SCREEN 2
COLOR1 = 1
X = 320
Y = 100
Pi = 3.14159
A2 = -.000001
```

```
Radius = 180
Total = 550
FOR I% = 1 TO 10
        READ Amt
        A1 = A2
        A2 = A1 - Amt / Total * 2 * Pi
        CIRCLE (X,Y), Radius,COLOR1,A1,A2
NEXT I%
DATA 10,20,30,40,50,60,70,80,90,100
END
```



注意：这些程序的格式和结构与前面那个非常类似。SCREEN 2 初始化屏幕。为了单色显示，Color 被置成 1。如果用户有合适的屏幕，可使用颜色。园心定于象素（320，100）上一X = 320，Y = 100。设定 Pi；各扇形的起始点设定稍偏中心一点，等于 -0.000001；半径置成 180。由于已知道总数为 550，因而将初始化为 550。如果不知道总数，可将它计算出。然后，用一个 FOR…NEXT 结构逐个画出 10 个扇形。

CIRCLE 语句的完整格式如下：

CIRCLE (x, y), Radius[, color[, start , end [, aspect]]]

其中： (x, y) 是圆心的位置，可用绝对或相对形式（用STEP）指定。

radius(半径）是数值表达式，确定圆的大小.

color(颜色）是一个整数表达式，确定圆的颜色，在中分辨率图形模式下，有效值为: 0～3（缺省值为3）；在高分辨率图形模式下，有效值为0和1（缺省值为1）。缺省颜色值是所在模式下最高的可用值。使用这些数值范围之外的颜色值将产生第5号运行时错误: Illegal Function Call（非法的功能调用）.

start(起始弧度）和end（结束弧度）是任选的数字参数，确定要绘制扇形的起始点和结束点，以弧度计。要仅画圆的一部分时，才使用这两个参数。如果Start和End被省去，则画整个圆。例如:

```
SCREEN 1
Pi = 4 * ATN (1)
CIRCLE (160, 100), 50, 3, 0, Pi
```

画一个从0弧度到Pi弧度（180度）的白色半圆。4 * ATN（1）以十六位的精度计算Pi（ATN是反正切函数）.

aspect(圆度）是一个数值表达式，决定圆的圆度或椭圆度。在中分辨率模下，缺省值为5／6（即0.833333）；在高分辨率模式下，缺省值为5／12（即0.416667）。取决于所用的特定显示板／视频监视器，用户可能需要调整这些值（或调整监视器），以获得更圆的圆。

§14.3.2.3 用LOCATE指令在图中标注元素

据我们所知，CIRCLE语句将画出其大小对应于所占总量百分比的馅饼图扇形块。然而，为更加清楚起见，馅饼图应标明各扇形块，给出它的值或百分比。为实现这个目的，可以在图形画出后用LOCATE指令将光标定位于期望打印的行和列；然后，打印标题或数字结果。CRT格式纸有助于标记计划打印数据的特定位置。

图14.6中的程序产生一个标记完整的馅饼图（如图14.7所示）.

LOCATE语句的格式为:

```
LOCATE [row][, column][, cursor][, start][, stop]]]
```

其中: row（行号）是一个整数表达式，确定光标将位于的屏幕行（1～25）。如果Help行（帮助行）用KEY OFF语句隐去了，才可以用第25行。

column（列号）指定光标将位于的屏幕列（1～80）。

cursor是一个数值；决定光标是否可见（0表示不可见；1表示可见）.

```
SCREEN 2
COLOR1 = 1
X = 320
Y = 100
Pi = 3.14159
A2 = -.000001
Radius = 180
TotalSales = 0
OPTION BASE 1
DIM DailySales(5)
FOR I% = 1 To 5
        READ DailySales (I%)
        LET TotalSales = TotalSales + DailySales (I%)
NEXT I%
FOR I% = 1 TO 5
        A1 = A2
        A2 = A1 - DailySales (I%) / TotalSales * 2 * Pi
        CIRCLE (X,Y),Radius,Color1,A1,A2
NEXT I%
DATA 250,200,150,300,100
LOCATE 7, 33 : PRINT "TUE"
LOCATE 8, 33 : PRINT "20%"
LOCATE 7, 47 : PRINT "MON"
LOCATE 8, 47 : PRINT "25%"
LOCATE 13, 24: PRINT "WED"
LOCATE 14, 24: PRINT "15%"
LOCATE 18, 40: PRINT "THU"
LOCATE 19, 40: PRINT "30%"
LOCATE 14, 54: PRINT "FRI"
LOCATE 15, 54: PRINT "10%"
END
```

图 14.6　画馅饼图的程序

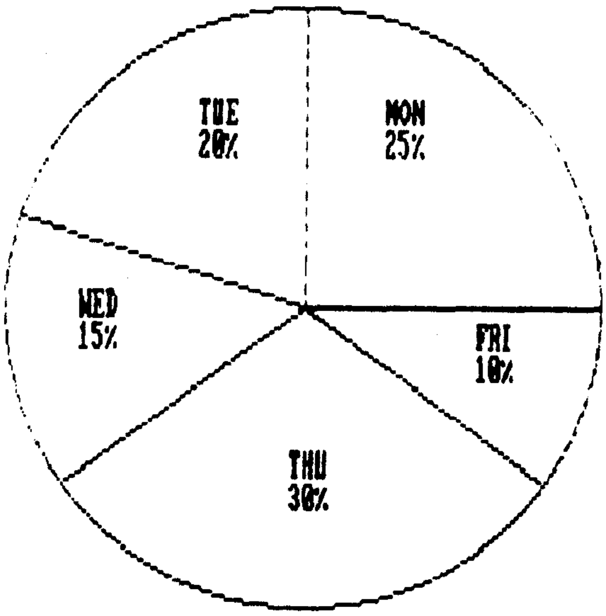


图 14.7　馅饼图

start 和stop 为整数表达式，决定光标的大小，并表示光标由多少个扫描行所组成。顶部扫描行是第 0 行，对于 color / Graphics Adapter （彩色/图形适配器）的底部扫描行为第 7 行；对于 Monochrome Adapter （单色适配器）则为第 13 行。

总之，LOCATE 语句常常在 PRINT 语句之前使用，确定屏幕上输出的位置，在本书的例子中，LOCATE 语句将与 PRINT 指令一起位于单一行中；两条指令用冒号分隔。

例:

```
LOCATE 9, 51
PRINT "A"
```

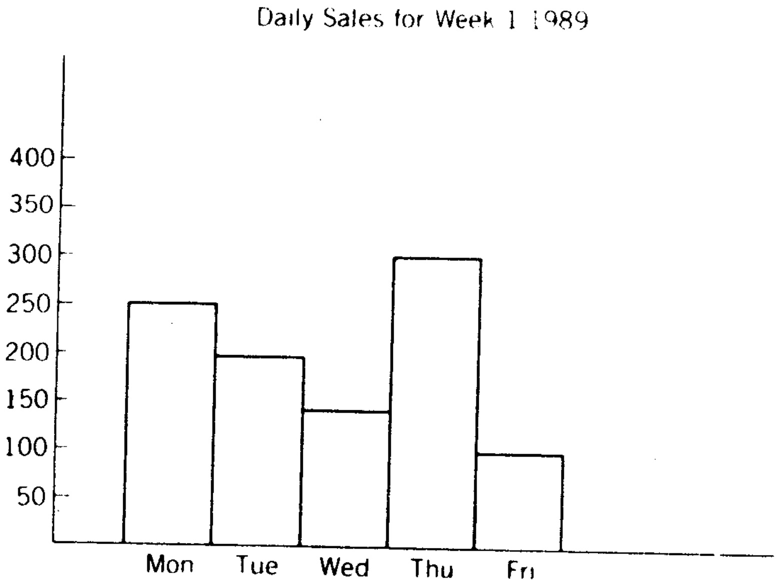
LOCATE 语句不仅用于图形，而且可用于显示用户友好人机界面的屏幕设计。

自我测验(一)

1. 大多数监视器每行显示________个正文字符，每屏显示________行.
2. ________是屏幕上用来创造字符和图形符号的独立点.
3. ________指令设定屏幕模式，以便于显示图形.
4. ________指令用来清屏.
5. ________是一种最适用于描述整体划分情况的图形.
6. (T或F) CIRCLE语句绘制带大小相等扇形的圆.
7. (T或F) SCREEN 语句可用于光标的定位，以便用文字标注图形.
8. (T或F) KEY OFF 语句用来隐去屏幕底部的帮助行.
9. 编写Turbo Basic 程序，绘制一个有六个扇形: 20%，15%，18%，15%，22% 和 10%的馅饼图.
10. 在第9题绘制的馅饼图中用A～F标记扇形块.

§14.3.3 直方图

如前所述，直方图可用来比较数量. 假设要显示如下的每日销售量:



§ 14.3.3.1 WINDOW 指令

WINDOW 指令可用来指定直方图所占的屏幕面积，其格式为:

```
WINDOW (x1, y1) - (x2, y2)
```

其中: (x1, y1) 和 (x2, y2) 为指定图形大小的座标。(x1, x2)及(y1, y2) 均为象素. 通常，将 WINDOW 指令设定为:

WINDOW (0, 0) - (数值的个数，最大值)

在给出的例子中，有五个数值. 每个数值对应于该周的某一天，星期四的数值是最大的,为 300. 因此，WINDOW 指令为:

WINDOW (0, 0) - (5,300)

在大数情况下，最大值和数值个数不是固定的，需要在编写该指令之前算出。

§ 14.3.3.2 LINE 指令

打印直方图的主要指令为 LINE 语句，其格式如下:

```
LINE [ (x1, y1) ] - (x2, y2) [, [color][, B[F]]
```

画线时，设定:

1. 起始和终止点

   (x1, y1) 为起始点; (x2, y2) 为终止点, x, y 坐标按象素给定.

2. color (颜色)

   待介绍COLOR语句时，再讨论有效的颜色选择.

3. BF—用色彩填充各方块.

通过编写:

```
LINE  ( )  -  ( ) , BF
```

可以略去 LINE 指令中的颜色设定. 两个逗号表示不使用颜色. 或者, 写成:

```
colorl=1
LINE  ( )  -  ( ) , colorl, BF
```

这是画馅饼图时用过的形式.

假设 Day (1) 到 Day (5) 包含每天的销售量, 则可以画出一个如下所述的的描述五个销售量的直方图.

```
FOR  I%-1 TO 5
LINE  (I%-1, O) - (I%, Day (i%)), BF
```

如前所述, 为使该程序可运行于任何屏幕, 略去颜色设定. 不过, 该程序段如本节所有程序一样, 要求计算机配有图形适配器.

输出打印结果如图 14.8 所示, 程序如图 14.9 所示(见下页).

§ 14.3.3.3 VIEW 指令

可以看出, 该直方图横跨屏幕的整个长度和宽度. 如果要想打印标注, 则需要作些修改, 可以用 VIEW 指令减少或缩小图形比例. 指出图形查看区域大小和位置的指令如下:

```
VIEW  (x1, y1) - (x2, y2)
```

其中: (x1, y1) - (x2, y2) 为按象素计的视口坐标. (0,0) 通常为屏幕的左上角. 例如, CGA 屏幕的右下角为 (199, 639); EGA 屏幕的右下角为 (349, 639).

VIEW 指令一般在 WINDOW 语句之后使用. WINDOW 语句指定实际用于画图的总体坐标, 而 VIEW 指令确定屏幕上图形出现的位置及其视口大小.

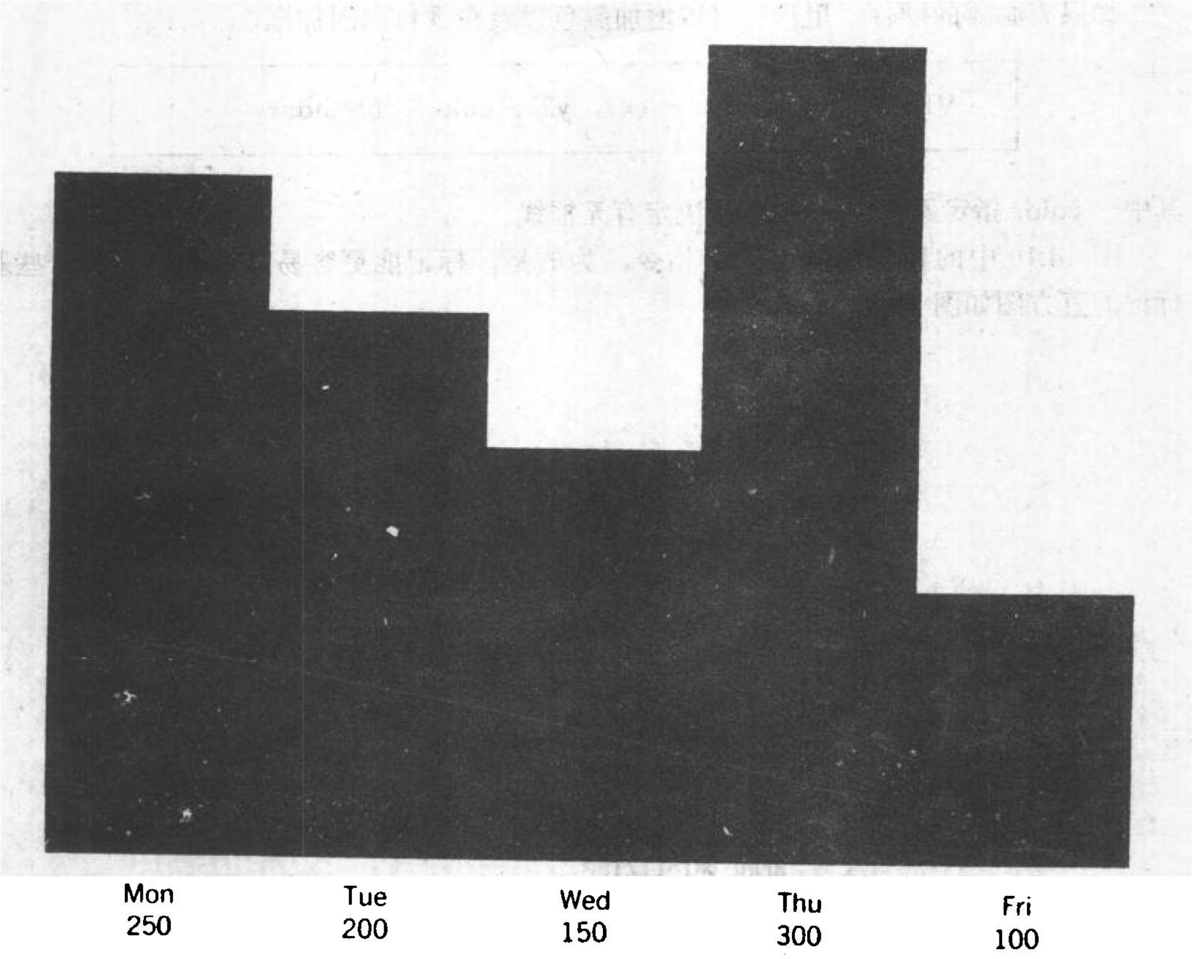


图 14.8　用 LINE 指令绘制的直方图

```
SCREEN 2
GOSUB GETDATA
WINDOW (0,0) - (5,300)
FOR I% = 1 TO 5
        LINE (I% - 1,0) - (I%,DAY(I%)),,BF
NEXT I%
END
GETDATA:
        OPTION BASE 1
        DIM DAY(5)
        FOR I% = 1 TO 5
        READ DAY (I%)
        NEXT I%
        DATA 250,200,150,300,100
RETURN
```

图 14.9　用 LINE 指令打打直方图的程序

如果有必要的硬件，用户还可以增加颜色以及在视口周围加框；

VIEW，(x1，y1) − (x2，y2)，color，boundary

其中：color 指定颜色，boundary 决定有无框线。

图 14.10 中的程序包括 VIEW 指令，为的是，标记能更容易加入图形，带一些基本标记的直方图如图 14.11 所示。

```
SCREEN 2
GOSUB GETDATA
WINDOW (0,0) - (5,300)
VIEW (165,1) - (315,99),,1
FOR I% = 1 TO 5
        LINE (I% - 1,0) - (I%, DAY(I%)),,BF
NEXT I%
END
GETDATA:
        OPTION BASE 1
        DIM DAY$(5)
        OPTION BASE 1
        DIM DAY(5)
        FOR I% = 1 TO 5
        READ DAY$(I%)
        READ DAY(I%)
        PRINT DAY$(I%),DAY(I%)
        NEXT I%
        DATA Mon,250,Tue,200,Wed,150,Thu,300,Fri,100
RETURN
```

图 14.10　带 VIEW 指令的直方图绘制程序

```
Mon        250
Tue        200
Wed        150
Thu        300
Fri        100
```

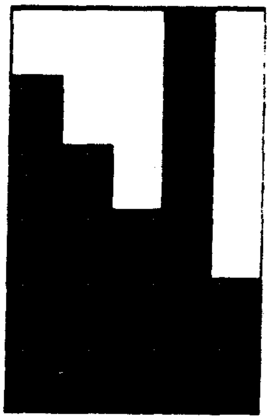


图 14.11　直方图

§ 14.3.3.4 LOCATE 语句

正如馅饼图一样，除非有明确标识各不同部分的标识，否则图形不是有意义的．至少，x 轴和 y 轴必须画出并标明．此外，图形名称也应标明；y 轴上的比例值应加入．

为了打印标记和数值，将 LOCATE 指令与 PRINT USING 指令联合使用．CRT 格式纸也有助于确定不同标记的打印位置．图 14.12 是打印直方图的完整程序．

```
SCREEN 2
GOSUB GetData
WINDOW (0,0) - (5,300)
VIEW (115,10) -  (180,70)
FOR I% = 1 TO 5
        LINE (I% - 1,0) - (I%,DAY(I%)),,BF
NEXT I%
LOCATE 3,1 : PRINT "MON"
LOCATE 3,9 : PRINT 250
LOCATE 4,1 : PRINT "TUE"
LOCATE 4,9 : PRINT 200
LOCATE 5,1 : PRINT "WED"
LOCATE 5,9 : PRINT 150
LOCATE 6,1 : PRINT "THU"

LOCATE 6,9 : PRINT 300
LOCATE 7,1 : PRINT "FRI"
LOCATE 7,9 : PRINT 100
END
GetData:
        OPTION BASE 1
        DIM DAY(5)
        FOR I% = 1 TO 5
                READ DAY(I%)
        NEXT I%
        DATA 250,200,150,300,100
RETURN
```

图 14.12　绘制直方图的完整程序

请记住，这些图例试图给用户提供一个用 Turbo Basic 准备图形的基本概况．要介绍图形产生指令及其选项，需要一整本书的篇幅．为此，如果用户对计算机绘图、建模或动画感兴趣的话，我们鼓励用户用本章概述中的指令和其它指令进行实践．

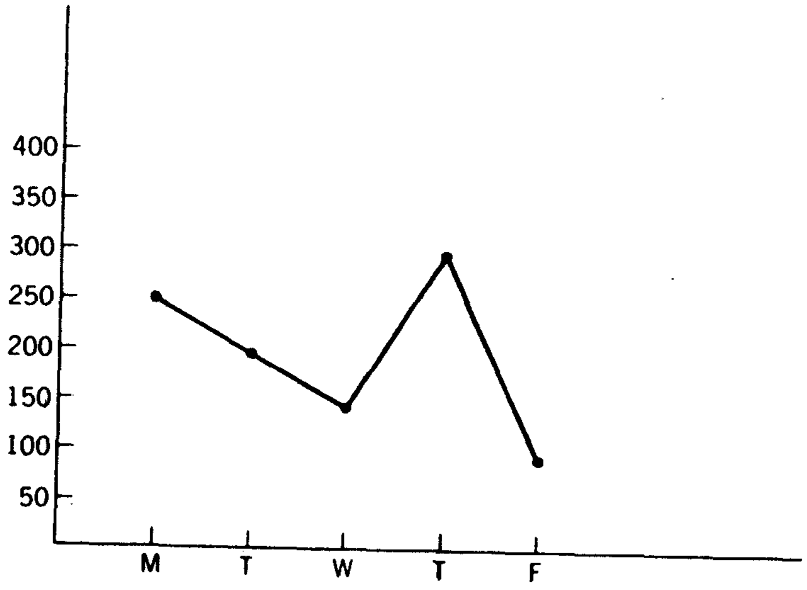
自我测验(二)

1. ________指令用来指定用于直方图的屏幕大小．
2. ________指令用来减少或缩小图形面积．
3. ________指令用来在屏幕上安放标记．
4. 什么是象素？
5. 用LINE 语句画一个正方形．
6. 用LINE 语句画一个长方形．

7. 用四个数据，150，225，75和110画一个直方图.
8. 修改第7题的直方图，使之位于屏幕的右上部.
9. 在屏幕的左上部增加标识：Week 1 Sales = 150，Week 2 Sales = 225，Week 3 Sales = 75 及 Week 4 Sales = 110.
10. 在图形正下方增加标注："WEETLY SALES"；并将x轴标作："WEEK"，Y轴标作："SALES".

§14.3.4 线条图

到目前为止，我们用日销售量已画过了馅饼图和直方图. 同样，也可以用如下所述的线条图描述日销售量.



线条图最适用于绘制趋势曲线. 在上图中，可以跟踪周一到周五的销售量趋势.

为了生成这样一个线条图，可以用 LINE 语句联接一系列点. 正如前面的实例所述，用 READ 和 DATA 语句存放日销售额. 这样，通过取消 INPUT 语句可简化程序. 实际上，应要求用户将它们输入，因为这些数据是变化的.

通常情况下，用户应首先用 CRT 格式纸设计图形的布局(参见图 14.13).

用如下 WINDOW 指令定义图形面积:

```
WINDOW (0, 199) - (319, 0)
```

下面两条指令用来画 x 轴和 y 轴:

```
LINE (56, 40) - (56, 180) ，画X轴
LINE (56, 40) - (256, 40), 画Y轴
```

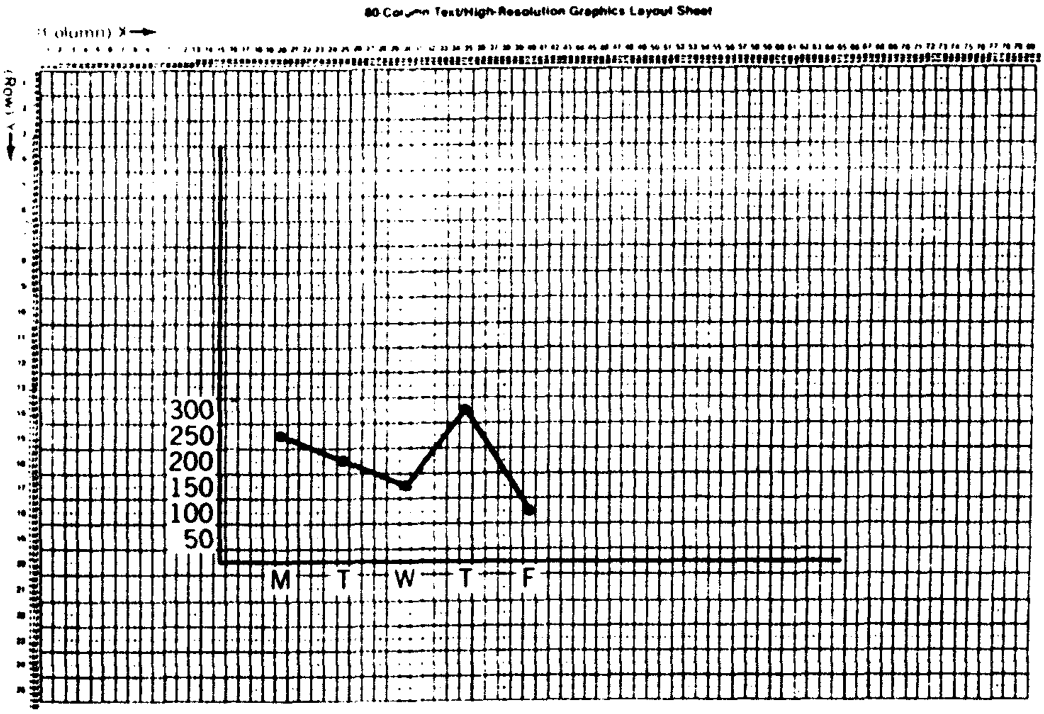


图 14.13 用 CRT 格式纸设计线条图

在线条图中，在各数值至下一个数值间画线，我们还将用小圆标明各个数值．因为线条图在各数值至下一个数值间画线，因而要用一个循环画从第一天销售量和第二天销售量、从第二天销售量到第三天销售量．从第三天销售量到第四天销售量，最后从第四天销售量到第五天销售量的线条．然后，用一个单独的圆画第五天的销售量（它没有另一点可连）．其基本格式为：

```
FOR I% = TO 4                          ' 画日期 1-2-3-4
    x1 = ···                           ' 计算 I 的 x1 坐标
    y1 = ···                           ' 计算 Total(I)的 y1 坐标
    x2 = ···                           ' 计算 I+1 的 x2 坐标
    y2 = ···                           ' 计算 Total(I+1) 的 y2 坐标
    CIRCLE (x1, y1), .2                ' 为 Total(I)画一个小圆
    LINE (x1, y1)    (x2, y2)          ' 从第I天的总数到第I+1天的总数连线
NEXT I%
CIRCLE (x2, y2), .2                    ' 为第五天的总数画一个小圆
```

最后，必须确定各组数值，如，(x1，y1) 和 (x2，y2) 的实际坐标，以便于画线．因为 I 轴始于象素 (56，40)，因此用横向坐标为 56 的象素作为基点．任意设定数值往右偏离起始点 20 个象素，从而有：

```
FOR I% = 1 TO 4
    x1 = 56 + 20 * I%
    y1 = ...
    x2 = 56 + 20 * (I% + 1)
    y2
    .
    .
    .
NEXT I%
```

y 轴也从象素（56, 40），因此用纵向坐标为 40 的象素作为基点．因为 y 值是从 100 到 300 变化的日销售量，因此用一个小数压缩各个日销售量．

```
FOR I% = 1 TO 4
        :
    y1 = 40 + .2 * Day(I%)
        :
    y2 = 40 + .2 * Day(I%+1)
NEXT I%
```

绘制基本线条图（不带图注）的程序如图 14.14 所示：绘制的线条图如图 14.15 所示．

```
SCREEN 2
GOSUB GETDATA
WINDOW (0,199)-(319,0)
LINE (56,40) - (56,180)
LINE (56,40) - (256,40)
FOR I% = 1 TO 4
        X1 = 56+ 20 * I%
        Y1 = 40 +.2 * DAY(I%)
        X2 = 56 + 20 * (I%+1)
        Y2 = 40  +.2 * DAY(I%+1)
        CIRCLE (X1,Y1),.2
        LINE (X1,Y1) -(X2,Y2)
NEXT I%
CIRCLE (X2,Y2),.2
END
GETDATA:
        OPTION BASE 1
        DIM DAY(5)
        FOR I% = 1 TO 5
                READ DAY(I%)
        NEXT I%
        DATA 250,200,150,300,100
RETURN
```

图 14.14　绘制基本线条图的程序

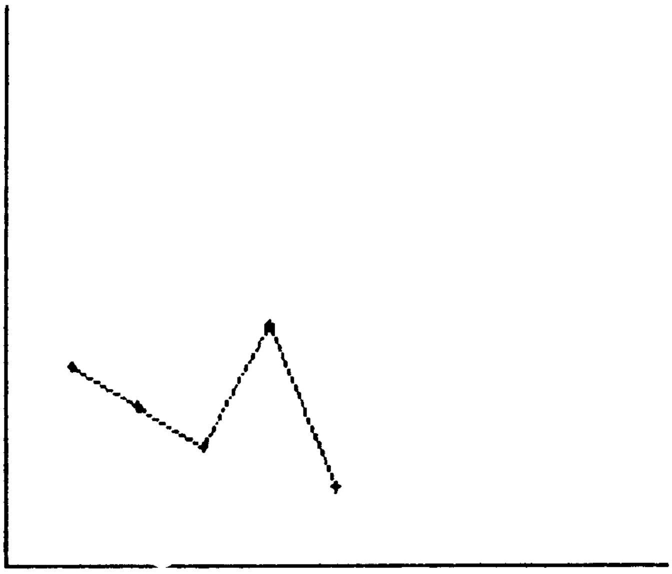


图 14.15　线条图

正如前面两个馅饼图和直方图的实例所述，我们认为应加入图注和数值方使图形可读。图 14.16 给出了完整的程序及其相应的图形。

```
SCREEN 2
GOSUB GetData
WINDOW (0,199) - (319,0)
GOSUB DrawLineChart
GOSUB AddLabels
END
GetData:
DIM Day(1:5)
        FOR I% = 1 TO 5
                READ Day(I%)
        NEXT I%
        DATA 250,200,150,300,100
RETURN
DrawLineChart:
        LINE (56,40) - (56,180)
        LINE (56,40) - (256,40)
        FOR I% = 1 TO 4
                x1 = 56 + 20 * I%
                y1 = 40 + .2 * Day(I%)
                x2 = 56 + 20 * (I% + 1)
                y2 = 40 + .2 * Day(I% + 1)
                CIRCLE (x1,y1),.2
                LINE (x1,y1) - (x2,y2)
        NEXT I%
        CIRCLE (x2,y2),.2
RETURN
AddLabels:
        LOCATE 14,7 : PRINT 300
        LOCATE 15,7 : PRINT 250
        LOCATE 16,7 : PRINT 200
        LOCATE 17,7 : PRINT 150
        LOCATE 18,7 : PRINT 100
        LOCATE 19,7 : PRINT 50
        LOCATE 22,20: PRINT "M"
        LOCATE 22,25: PRINT "T"
        LOCATE 22,30: PRINT "W"
        LOCATE 22,35: PRINT "T"
        LOCATE 22,40: PRINT "F"
RETURN
```

图 14.16(a)　绘制线条图的完整程序

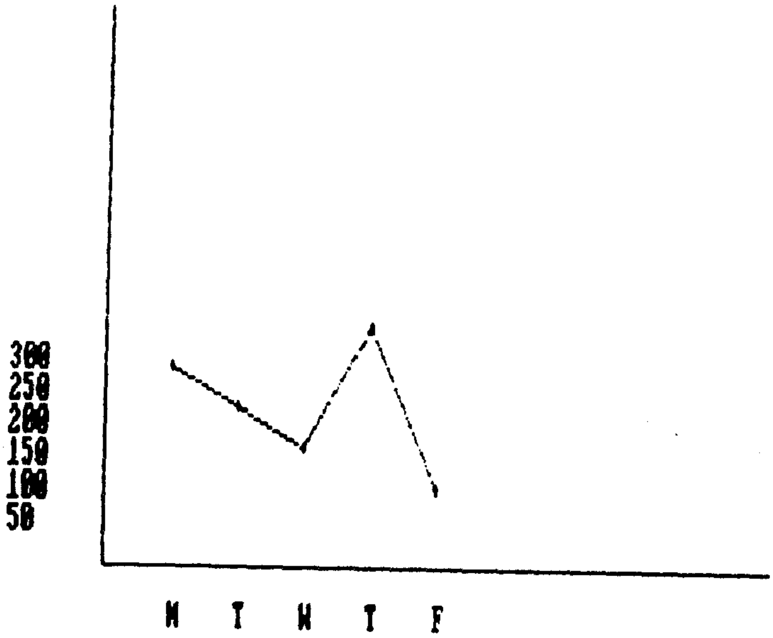


图 14.16(b)　绘制的线条图

自我测验(三)

1. (T或F) 线条图不适合于描绘趋势.
2. (T或F) 线条图最适用于描绘整体的划分情况（百分比).
3. ________语句可用来将光标定位于第10行，第15列.
4. ________语句指定画线所需的x和y坐标的象素值.
5. 编一条自象素（15，30）到象素（50，75）画线的语句.
6. 写出在第5题画出的线条两端画小圆的语句.
7. 写出画一个三角形的语句.
8. 写出画一个方块的语句．要求有两条对角线穿过.
9. 绘制一个线条图，标出下述各点:

100, 130, 160, 190, 110, 140

10. 假设第9题绘制的线条图是为某珠宝店的一周销售数目而设计的，要求在图中加入相应的标注.

§14.3.5 COLOR 指令

图形显示如果配有颜色，则直观得多. 为了编写带色彩的图形显示程序，或使用彩色图形软件包，必须要有一个彩色屏幕和一个带有接收颜色代码所必需的电路和彩色图形板. 若要打印彩色图形，必须有彩色图形打印机. 使用 COLOR 语句时，用户必须先用 SCREEN 语句作为开始.

为了屏幕上显示彩色正文或图形，使用下述 COLOR 指令:

```
COLOR [background][, palette]
```

其中: background (背景色) 和paleete (调色板) 为0～31的数字. 方括号[ ]表示任选条目. 因此, background 和 palette 两条目均为任选项，即 COLOR 指令只用来修改背景色或前景色 (调色板). 下面为 COLOR 指令可能使用的颜色码.

单色显示器上有效的前景色		彩色／图形监视器上有效的前景色和边框色			
0	黑	0	黑	8	灰
1	带下划线字符	1	兰	9	浅兰
2～7	白／淡黄／绿	2	绿	10	浅绿
		3	青	11	浅青
(背景或边框可以是白色或黑色)		4	红	12	浅红
		5	深红	13	浅棕色
		6	棕	14	黄
		7	白	15	高亮度白
		(单色显示时，背景色不能用 0～7)			

将前景色设为 16 加上表中任意数字，可实现闪烁显示. 因此，无论在屏幕上显示什么，从 16 到 31 的任一前景颜色号都将其闪烁. 注意，彩色监视器前景色安排成: 将任一"标准"颜色加上 8 便生成该颜的高亮度形式. 此外，还应注意，青色是兰绿色.

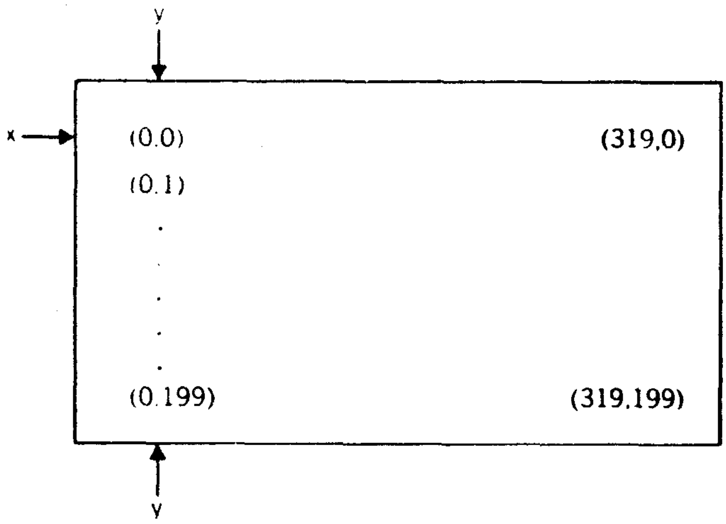
§ 14.3.6 用于画线的 PSET 指令

PSET 或"Point Set （点设定）"指令用来给象素的定位，并在所指定的位置上画点．因而，它类似于 LOCATE 指令。不过，它实际上画点而不仅仅给光标定位．为指定象素的位置，PSET 指令象 LOCATE 一样，要求两个数字或座标值:

```
PSET    (X, Y)
```

x, y 的上限随所有屏幕的类型而变化．对于中分辨率图形模式，x 一般为 0～319 间的数字，y 通常为 0～199 间的数字；对于高分辨率图形模式，x 一般为 0～639 间的数字，y 通常为 0～199 间的数字．

中分辨率



PSET 在其坐标标定的位置上显示一个小点．为了画一幅由一系列点构成的图形，PSET 指令一般与 FOR…NEXT 联合使用．用这种方法，循环产生一系列可用来画线、圆、甚至任意图形的点．

为了画出一条类似于用 LINE 指令显示的线条，编程如下:

```
SCREEN 2
FOR Y = 8 TO 160
      PSET (80, Y)
NEXT Y
```

执行 RUN （运行）命令后，将打印一条实线．

PSET 也可以用于颜色图形，它的完整格式为:

```
PSET (x, y) [, attribute]
```

带方括号的条目[, attribute (属性)]为任选项。若存在该项，逗号不可省。注意，attribute 可能是一种颜色。

坐标值可用来指明特定的 x 和 y，或指定与最近引用点有关的点。PSET STEP (10, 10) 将自前面指定点处下移 10 个象素，并横跨 10 个象素。

用 PSET 语句画一个方块

我们也可以在 FOR…NEXT 循环中联合使用 FOR…NEXT 和 PSET 指令画方块，下面为一个实例：

```
SCREEN 2
FOR I% = 10 TO 100
   FOR I% = 10 TO 200
       PSET (I%, J%)
   NEXT J%
NEXT I%
```

如果用户键入并运行该程序，则可以看到实际画出的方块是一个黑底白筐，用户可通过在 PSET 指令中增加[, color] 项的方法来改变颜色。

§ 14.3.7 DRAW 指令

DRAW 语句可用来获取图形或任何特殊图形符号，其格式如下：

```
DRAW string
```

其中，string (字符串) 是一组如下所述的移位命令：

移位命令	功能 (n 为象素的数目)
Un	上移 n 个象素
Dn	下移 n 个象素
Ln	左移 n 个象素
Rn	右移 n 个象素
En	向左上方移动 n 个象素
Fn	向左下方移动 n 个象素
Gn	向左下方移动 n 个象素
Hn	向左上方移动 n 个象素

DRAW 语句中的整个字符串需用引号括括住.

图 14.17 为上述移位命令的示意图:

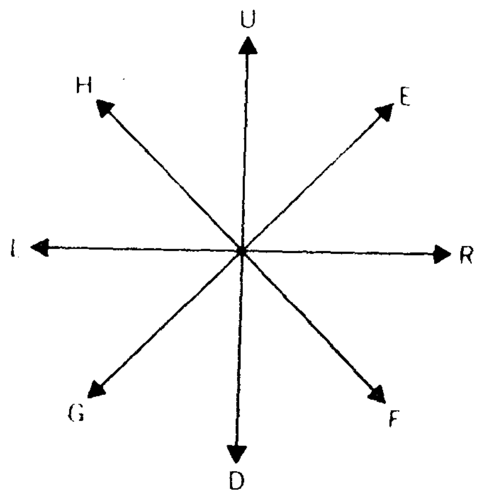


图 14.17　移位命令

下述命令在中分辨率图形模式下 (SCREEN 1) 画一个正方形:

```
DRAW "R60 D50 L60 U50 "
```

下面命令在高分辨率屏幕模式 (SCREEN 2) 下画一个正方形:

虽然 DRAW 指令中有许多其它选项可供使用, 但上述命令可以画一些非常有趣的图形. 在此, 重复一遍, 实践是学会如何有效使用 DRAW 指令的最好方法.

SCREEN 1 模式下的图形

为画一个三角形:

```
SCREEN 1
PSET (60, 125)
DRAW "E100; F100; L19"
```

DRAW 语句中分号可有可无.

为画一个方块, 有:

```
SCREEN 1
DRAW "U40; R40; D40; L40"
```

自我测验(四)

1. ________指令用来确定象素的位置，并在那个位置上画点。
2. 相关位置可能用________指令指定。
3. (T或F) PSET 和DRAW 指令可用来显示线条和方块。
4. (T或F) DRAW 指令指定一串给光标定位和移位的数值。
5. ________指令可用于获取图形。
6. (T或F) SCREET 2 命令用来指定中分辨率图形模式。
7. 在红底上画一个兰色三角形。
8. 用单条语句在黑底上画一个白色方块。
9. 用DRAW 和PSET 指令画四个三角形，组成正方形。
10. 画一个方块，其中两条竖线为一种颜色，两条横线是不同的颜色。

§14.4 用 Turbo Basic 产生声音

一条称作 BEEP 的简单 Turbo Basic 指令可产生一个持续四分之一秒的声音。万一发生错误或某动作随即进行时，它可用来给用户提供一个信号。

通过使用 Turbo Basic 中的 SOUND 语句，可产生不同频率的声音。用户也可以用计算机的内部时钟设定声音的速度或节拍。SOUND 指令的格式如下:

```
SOUND frequency , duration
```

其中: frequency （频率）是用户赫兹表示的声音频率，即每秒的周期数。其数值范围为 37～32767.

duration(持续时间）是声音持续的时间长度，单位为: 1/18.2 秒。因此，duration 为 18 时，设定持续时间大约为 1 秒的音调。其数值范围为: 0～65535

图 14.18(见下页) 给出了音符表及其相应频率。注意，中音 C 的频率为 523.25 赫兹。比图 14.18 给出的频率更低和更高的频率也是可能的。

正如 DRAW 指令使图形显示更加容易一样，PLAY 指令也同样使音符更容易产生。PLAY 指令的格式如下所述:

```
PLAY String expression
```

其中: String expression （字符串表达式）的值如图 14.19 所示:

Note	Frequency	Note	Frequency	Note	Frequency
C	130.81	C	523.25	C	2093.00
D	146.83	D	587.33	D	2349.40
E	164.81	E	659.26	E	2637.00
F	174.61	F	698.46	F	2793.80
G	196.00	G	783.99	G	3136.00
A	220.00	A	880.00	A	3520.00
B	246.94	B	987.77	B	3951.00
C	261.63	C	1046.50	C	4186.00
D	293.66	D	1174.70	D	4698.80
E	329.63	E	1318.50	E	5274.00
F	349.23	F	1396.90	F	5587.60
G	392.00	G	1568.00	G	6272.00
A	440.00	A	1760.00	A	7040.00
B	493.88	B	1975.50	B	7902.00
BASS		MIDRANGE		TREBLE	

图 14.18　音符表及其相应的频率

字符串值	意　义
A～G	演奏指定的音符；指定音符之后的“#”或“+”升高音调，而“-”降低音调.
On	设定八音度；第三个八音度从中 C 开始．缺省值为第四个八音度.
>N	升高一个八音度，并演奏音符 N.
<N	降低一个八音度，并演奏音符 N.
Nn	演奏第 n 号音符(n 必须是 0～84 间的数).
Ln	为后面的音符设定时间，时间为 1／n(n 必须在 1～64 之间).
Pn	休止，休止时间为 1／n(n 必须在 1～64 之间设定速度).
Tn	设定速度；n 表示每分钟四分音符的数目，必须在 32～255 之间.
MF	用前台“方式”演奏
MB	用“后台”方式演奏
NM	正常演奏，每个音符按指定时间的 7／8 演奏.
ML	音乐连奏，每个音符按满的指定的时间

	演奏.
MS	断续演奏，每个音符按指定时间的 3／4 演奏.
X 变量	执行字符串变量中的命令.

图 14.19　PLAY 指令字符串表达式的值

图 14.20 用 PLAY 指令演奏一段假日曲调，并显示一颗圣诞树.

```
N = 15
CLS
LOCATE 7, 1
PRINT
PRINT TAB(N); "               *"
PRINT TAB(N); "              ***"
PRINT TAB(N); "             *****"
PRINT TAB(N); "            *******"
PRINT TAB(N); "           *********"
PRINT TAB(N); "          ***********"
PRINT TAB(N); "         *************"
PRINT TAB(N); "              *****    "
PRINT TAB(N); "              *****    "
LET S$ = "-------------------------Happy Holidays"
PLAY "MBT25502"
PLAY "G2.F4E2D2C2D2E2C2DEFDE2D4C4.01B202C2.P4"
PLAY "G2.F4E2D2C2D2E2C2DEFDE2D4C4.01B202C2.P4"
PLAY "E6F6G6.A6B603C6.02B2A2G2"
PLAY "G2.F4E2D2C2D2E2C2DEFDE2D4C4.01B202C2.P4"
'*********************************************
' Flashes Happy Holidays across the bottom
'         of the screen
'*********************************************
FOR I% = 1 TO 15
        LOCATE 20, 13
        PRINT STRING$(LEN(S$) - I%, "-"); MID$(S$, 41 - I%, I%); "-"
NEXT I%
FOR I% = 2 TO LEN(S$)
        FOR J% = 1 TO 100
        NEXT J%
        LOCATE 20, 23
        PRINT "-"; MID$(S$, I%, LEN(S$)); STRING$(I%, "-")
NEXT I%
LOCATE 20, 23
PRINT "-"; STRING$(LEN(S$), "-")
```

图 14.20　产生假日曲调的 PLAY 指令

§ 14.5　结论

本书重点在于 Turbo Basic。我们自始至终使用的编程方法是结构化程序设计方法.

用户将发现许多 Turbo Basic 程序也可以用其它版本的 Basic 运行. 虽然它们运行起来不如 Turbo Basic 快，且未充分利用窗口环境，但 Turbo Basic 的语法与其它版本的 Basic 的非常类似，使之兼容性很好.

下面给出 Turbo Basic 与其它版本 Basic 相比所具有的许多优点的最终小结. 所有这

些优点均在本书中作了介绍。

Turbo Basic 特性小结

- 提供上下文有关的帮助
- 完全支持递归
- 具备定制用户界面和编辑器的能力
- 为字符串提供充足的64K内存
- 标准的IEEE浮点格式
- 支持8087 （数学协处理器）
- 程序的大小只受计算机保用内存的限制（而不是64K的限制）
- 支持EGA和CGA
- 支持IMB PS／2和MCGQ；在640K×480的分辨率下，可以使用2种和16种颜色。
- 编译器，编辑器，运行时程序库和可执行程序的完全集成；具有独立的编辑信息、跟踪、连接程序库、用户界面和执行窗口。
- 发生编译、运行时和I／O错误时，在源程序中指出错误发生处。
- 可访问局部、静态和全局变量
- 新的长整型（32位）。
- 下拉选单
- 全窗口管理

§14.6　本章辅导

§14.6.1 本章小结

一. Turbo Basic 图形概述

1. 一幅插图、图片或曲线图通常比文字输出更有意义、更有用。
2. Turbo Basic 可生成任何形式的图画，统计图或动画。
3. 一些软件包可产生图形。
4. 商业中最常用的图形类型为馅饼图，直方图和线条图。

二. 显示图形所需的微型设备

1. 为显示图形，需要一台图形监视器和一个图形适配器.
2. 监视器的类型:
   a. 大多数监视器每行显示80个正文字符，每屏显示25行.
   b. 中分辨率屏幕一般可为640×200个象素编址.
   c. 高分辨率屏幕一般可分640×350个象素编址.
   d. Turbo Basic 支持单色、CGA、EGA、MCGA、和VGA图形适配器.

三. Turbo Basic 图形指令

1. SCREEN语句用于设定屏幕显示模式.
2. 图形应先在CRT格式纸上设计.
3. 馅饼图用于描述整体的划分情况.
   a. CIRCLE 语句用于画圆. 为了将圆分成扇形，应用公式计算正确的角度.
   b. LOCATE指令用于标注图形元素.
4. 直方图用于数值比较.
   a. WINDOW 语句指定用于直方图的屏幕大小.
   b. LINE 指令用于画直方图中的各方块.
   c. VIEW 指令用于缩小图形.
   d. 再次用LOCATE 指令标注标记.
5. 线条图最适合于描绘趋势.
   a. LINE 语句用于连接一系列点.
   b. 各点用CIRCLE 语句确定.
   c. WINDOW 语句用于确定图形大小.
6. COLOR 语句用于设定图中颜色.
   a. 彩色屏幕和彩色适配器是生成彩色图像所必不可少的.
   b. 前景或背景可以使用36种颜色.
7. PSET 语句用于在指定象素处画点.
   a. 点可联成线
   b. 该命令可用来绘制各种类型的图形.
8. DRAW 语句用于绘制图片、统计图或特殊图形符号.
   a. 该语句指定画线的方向和象素数目.
   b. PSET 语句用于设定图形的起始点.

四. 用 Turbo Basic 产生声音

1. BEEP指令使计算机产生一个持续四分之一秒的声音.
2. SOUND 语句设定声音的频率和持续时间.
3. PLAY 指令是一个产生音乐的简易方法.

§ 14.6.2 本章自测

1. (T或F) 在高分辨率图形模式下，可编址的象素数比中分辨率图形模式下的少.
2. (T或F) 馅饼图最适合于描述整体的划分情况.
3. (T或F) 图形大小可用VIEW 语句调整.
4. (T或F) LOCATE 可用来指定起始绘制点.
5. Turbo Basic 彩色图形可使用________种不同的颜色.
6. 使用高分辨率图形时，要用________指令.
7. ________指令使计算机产生个响声.
8. 使用________指令时，可能要观察实际绘制的图形.
9. (T或F) CIRCLE 指令可包含颜色，也可略去.
10. 在中分辨率模式下，编程绘制一个8×8的国际象棋棋盘.

§ 14.6.3 实例

这个程序用于从键盘上接收五年销售信息中的各销售量. 要求在屏幕的一半上显示线条图，在另一半上显示同一数据的直方图；两个图形均应标记.

图 14.21 给出了程序和人机对话的过程. 层次图如图 14.22 所示

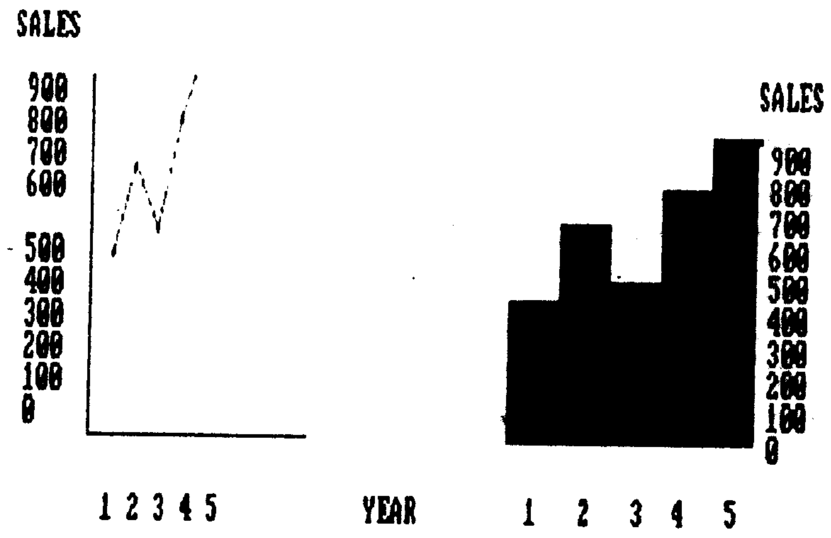


图 14.21(a) 运行结果样图

```
SCREEN 2
GOSUB GetData
GOSUB LineChart
GOSUB BarChart
END
GetData:
     DIM Sales (1:5)
     FOR I% = 1 To 5
          PRINT "Enter sales for year ";I%;"_
                  (enter numbers between 100 and 1000)"
          INPUT Sales (I%)
     NEXT I%
RETURN
LineChart:
     WINDOW (0,199) - (319,0)
     VIEW (10,40) - (220,150)
     LINE (56,40) - (56,200)
     LINE (56,40) - (256,40)
     FOR I% = 1 To 4
          x1 = 56 + 20 * I%
          y1 = 40 + .2 * Sales(I%)
          x2 = 56 + 20 * (I% + 1)

          y2 = 40 + .2 * Sales (I% + 1)
          CIRCLE (x1,y1),.2
          LINE (x1,y1) - (x2,y2)
     NEXT I%
     CIRCLE (x2,y2),.2
     LOCATE  4,1 : PRINT "SALES"
     LOCATE 16,1 : PRINT 0
     LOCATE 15,1 : PRINT 100
     LOCATE 14,1 : PRINT 200
     LOCATE 13,1 : PRINT 300
     LOCATE 12,1 : PRINT 400
     LOCATE 11,1 : PRINT 500
     LOCATE  9,1 : PRINT 600
     LOCATE  8,1 : PRINT 700
     LOCATE  7,1 : PRINT 800
     LOCATE  6,1 : PRINT 900
     LOCATE 19,28 : PRINT "YEAR"
     LOCATE 19,7  : PRINT 1
     LOCATE 19,9  : PRINT 2
     LOCATE 19,11 : PRINT 3
     LOCATE 19,13 : PRINT 4
     LOCATE 19,15 : PRINT 5
RETURN
BarChart:
     WINDOW (0,0) - (5,999)
     VIEW (300,40) - (450,130)
     FOR I% = 1 TO 5
          LINE (I%-1,0) - (I%,Sales(I%)),,BF
     NEXT I%
     LOCATE 19,39 : PRINT 1
     LOCATE 19,43 : PRINT 2
     LOCATE 19,47 : PRINT 3
     LOCATE 19,50 : PRINT 4
     LOCATE 19,54 : PRINT 5
     LOCATE 6,57  : PRINT "SALES"
     LOCATE 17,57 : PRINT 0
     LOCATE 16,57 : PRINT 100
     LOCATE 15,57 : PRINT 200
     LOCATE 14,57 : PRINT 300
     LOCATE 13,57 : PRINT 400
     LOCATE 12,57 : PRINT 500
     LOCATE 11,57 : PRINT 600
     LOCATE 10,57 : PRINT 700
     LOCATE  9,57 : PRINT 800
     LOCATE  8,57 : PRINT 900
RETURN
```

图 14.21(b) 程序

```
Enter sales for year 1 (enter numbers between 100 and 1000)
? 400
Enter sales for year 2 (enter numbers between 100 and 1000)
? 600
Enter sales for year 3 (enter numbers between 100 and 1000)
? 450
Enter sales for year 4 (enter numbers between 100 and 1000)
? 700
Enter sales for year 5 (enter numbers between 100 and 1000)
? 850
```

图 14.21(c) 运行过程

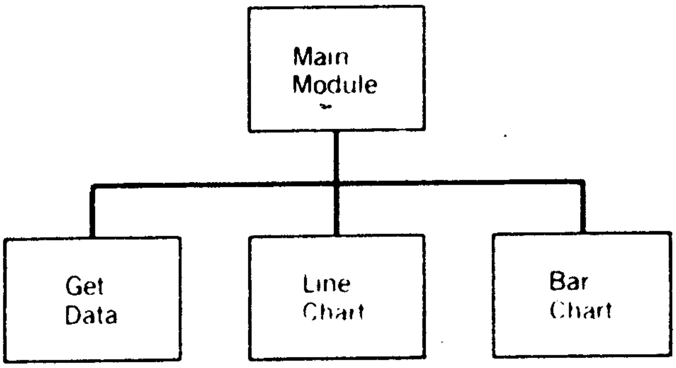


图 14.22 层次图

§ 14.6.4 复习题

一. 综合题

1. 给定每周的输入数据：250.00，325.00，275.00，335.00，365.00．画出一个带相应标记的线条图．
2. 用第1题中的数据画一个带相应标记的直方图．
3. 用第1题中的数据画一个带相应标记的馅饼图．
4. 编一小段程序，实现一无论什么时候输入0，计算机都响一声．
5. 编一程序使计算机演奏C，D，E 和F 音符．
6. 编写一个在圆内画三角形的程序．如果有彩色适配器．为圆和三角形配不同颜色．
7. 画一个带15%，25%，30%，20%和10%的扇形的馅饼图．
8. 编程画一个长方形，并在其中加入“GRAPHICS”字样．

9. 下列坐标哪个不可能在IBM 微型机中出现?

a. 540, 5
b. 15, 382
c. 283, 4
d. 650, 84

10. 下列哪条指令用于选择一个标准的高分辨率屏幕?

a. SCREEN 1, 2
b. PSET
c. HIEGRES
d. SCREEN 2

11. 下面语句将画出什么形式的线条?

```
LINE  (100, 100) - (400, 100), 1
```

a. 一条横线。
b. 一条竖线
c. 将从右到左的斜线
d. 一条从左到右的斜线

12. (T或F) "LINE (100, 50) -() 200, 150), 1"语句将画一条横线.

13. (T或F) 下面程序将画一个宽20个象素，高80个象素的长方形

```
FOR I% =1 TO 20
    LINE = (185, 150-I
)- (265, 150-I%), 1.
NEXT I
```

14. (T或F) 下述语句画一个整圆.

```
CIRCLE (160, 100); 50, 1, 2 2 * PC, 1
```

二. 数据验证

修改实例，使之包括如下功能代码: (1) 测试所有错误; (2) 打印一个总数控制表 (处理的记录数、遇到的错误和分类总数).

§14.6.5 调试练习

修正如下程序:

1.
```
SCREEN 1
LOCATE 34, 23
PRINT "Turbo Basic"
END
```
2.
```
SCREEN 2
CIRCLE (100, 320), 110
END
```
3.
```
SCREEN 2
WINDOW (199, 0) - (0, 319)
VIEW (50, 100) - (150, 150)
LINE (60, 100) - (110, 130), B
END
```
4.
```
SCREEN 2, 1
COLOR 1, 4
LOCATE 13, 35
PRINT "COLOR"
END
```

§ 14.6.6 编程题

1. 编写一个演奏歌曲"Happy Birthday To You （祝你生日愉快） "的Turbo Basic 程序，并在演奏时将"HAPPY BRITH DAY "显示在屏幕上.
2. 编写一个先显示"COMPUTERS ARE FUN"，接着暂停，然后清除屏幕并响一声的程序.
3. 编写一个为如下给出的每月销售数据绘制直方图的程序，并加标注.

January	$ 15.000
February	$ 19.750
March	$ 22.500
April	$ 15.500
May	$ 18.500
June	$ 24.500
July	$ 28.500
August	$ 32.2500
September	$ 31.000
October	$ 27.500
November	$ 29.000
December	$ 32.500

4. 编写一个程序以馅饼图形式描述第3题中的数据，并加标注.
5. 编写一个程序用第3题中的数据产生一个加标注的线条图.
6. 编程为下面列出的每日温度画直方图，并加上相应的标注.

Monday	50°F
Tuesday	64°F
Wednesday	80°F
Thurday	60°F
Friday	70°F
Saturday	62°F
Sunday	55°F

7. 编程为上题绘制加标注的线条图。
8. 编程为六名销售人员各自的酬金画馅饼图.

第一位售货员	$ 250
第二位售货员	$ 330
第三位售货员	$ 250
第四位售货员	$ 300
第五位售货员	$ 500
第六位售货员	$ 190

参考答案

第　一　章

自我测验(一)

1. 机器
2. 符号；机器语言非常复杂（它们使用冗长的指令码，而且程序员必须为存放数据指定实际机器存储单元或地址）.
3. 翻译（编译或解释）.
4. 源或应用
5. 编译器；更有效的目标码
6. 翻译程序（编译器或解释器）；目标或机器语言
7. 语法；在符号程序设计语言的使用过程违反了程序设计规划
8. 结构化程序的走查
9. 执行或运行；为进一步处理，以翻译后或目标码的形式保存.
10. 归档（编写文档）

本章自测

1. 输入
2. 输出
3. 磁盘；键盘
4. 结构化程序设计；自顶向下程序设计
5. T
6. 解释器；编译器
7. 编译器
8. T
9. 软件
10. 操作系统
11. F—计算机程序应在编码之前设计
12. 伪代码；层次图

13. 源；目标
14. F—编译器检查语法，不检查逻辑错误
15. Beginner's All-Purpose Symbolic Instruction Code.
16. T
17. 模块
18. F—在Turbo Basic中行号可有可无.
19. PRINT；LPRINT
20. 当前程序显示在Edit窗口中.
21. 所有从键盘键入的数据和PRINT操作的结果一起显示在RUN窗口中.
22. 程序开发过程中的步骤为:
    a. 获取程序指标
    b. 用规划工具设计程序
    c. 编码并键入程序
    d. 将程序翻译成机器语言
    e. 测试程序
    f. 编写文档
23. DO WHILE；LOOP
24. 终止值；Amt大于或等于9999
25. F—为帮助设计程序中的逻辑结构应先准备.

第 二 章

B188-32 ②

自我测验(一)

1. micro computer
2. 个人计算机 (personal computer)
3. A
4. 操作系统；直接开机启动
5. T
6. 回车
7. DOS，OS/2
8. 日期，时间
9. copy (如果目标盘已格式化)或 diskcopy
10. T

六章自测

1. PC; PS/2
2. 5*F1/4 ; 3*F1/2
3. 3*F1/2 ;
4. T
5. C
6. T
7. data; time
8. 提示; A
9. copy *.* C:
10. 命令.
11. 用光标移动键将所要求的命令加亮，然后回车，按所要求命令的第一个字母.
12. Edit; 双线框
13.
    a. 选择File命令中的"WRITE TO"选项
    b. 给出新文件名提示后，输入Testit.
    c. 按回车键将文件名传送给计算机.
    d. 按Esc键退出File命令.
    e. 键入字母E进入Edit命令（或将Edit命令加亮，然后按回车键.）
    f. 开始输入程序.
14. 在编辑状态用F2功能键，或选用File命令中的save选项.
15. F1功能键.
16. keyboard break ON; Option; Ctrl-Break.
17. 采用如下三种方法之一:
    a. 用光标移动键将光标移至i的右端，然后用BackSpace键删除i，键入a;
    b. 用光标移动键将光标移至i，按Del（或Delete）键删除i，然后插入a;
    c. 用光标移动键将光标移至i的位置，按INS（Insert）键进入替换状态，键入a以替代i.
18. 用Home键或←光标移动键将光标移到INPUT行的左端，按回车键，则在INPUT行和PRINT行中出现一个空行，用↑光标移动键将光标移到空行.
19. 将光标移到S的位置，触发Ins（或Instet）键，进入替换状态，按字母n使其替代s(这是唯一的改错方法).
20. 将光标移到S的位置，触发Ins（或Instet）键，当Insteten在状态行消失，键入U.
21. "乒乓"开关键具有如下特性，按一次时某种功能被起动，而按第二次时这种功能被取消，第三次按下时这种功能再次被起动,如此下去. 如Ins（Insert）键，CapsLock键等为"乒乓"开关键.

18A8-32(2)

22. F 首先执行编译，然后运行。
23. T
24. F2将程序存盘，Esc退出Edit命令状态。
25. T

第 三 章

自我测验(一)

1. 变量；常量
2. 结束；退出循环
3. a和c正确。下面是错的：
    b. INPUT 是保留字。
    d. Name　Of　Company $ 是字符串变量。
    e. Test　Score 有一个空格。
    f. 9th.Inning.Scone 以数字开头。
4. b正确。下面的不对：
    a. $ Address没有以字母字符开头，且没有以 $ 结尾。
    c. Name.In 没有以 $ 结尾。
5. b、c、d合法。下面的不对：
    a. 123.45+在最左边没有+或一号。
    e. 1，000 有一个逗号，逗号不是合法的数字字符。
    f. $ 23.00 有美元符号，这是非法的数字字符。
6. a对，b错误。因"Error　Message　123必须以双引号结尾。
7. T。
8. 问号？（或提示）
9. 数字变量；字符串变量；读入一个名字和数量，加10到这个数量上，然后打印名字和新数量。
10. 数字常量。

自我测验(二)

1. 数字变量
2. T。Turbo　Basic是大小写不敏感的语言。
3. 两个；逗号
4. 问号？或提示；数据或输入
5. Enter（或　或回车Return）

6. 12-是非法的数字输入（-12才对）；同样，ABC含有字母，也不是合法的数字输入。
7. 结构化
8. 输入到Length的值等于或小于0．DO WHILE Length＞0表示到LOOP间的指令集在 Length 大于 0 时被执行．
9. T．第一条输入INPUT读入初始值；而DO WHILE … LOOP处理这些值，并读入其余的值，直至再没有数据为止．
10. T．
11. WASHINGTION（输入语句中的逗号结束到变量EmpName$ 的传送）．
12. WASHINGTON，GEORGE（问号保证整个值，包括逗号传送给变量．）
13. 空串，无值。
14. ABC（无引导空格）
15. b̸b̸ABC
16. 在（a）中，要求一行输入两个数字值并用逗号分隔开；
    在（b）中，要求用两行输入，一行只有一个数字值．
17. 有语法错误．因为字符串操作数（Amt$）不能用在有数字操作数（Total和500）的算术运算中，既使 Amt$ 含有数字值．
18. 有．因为Code$ 为一字符串变量，必须用字符串值赋值．
19. 有．HoldIt是一个数字变量，应该被赋予一个数字值．语句应为Holdit＝123或Holdit＝"123"．
20. 有．Name$ 是保留字．

自我测验(三)

1

伪代码

```
START
    读入加仑数和行驶的英里数
    DO WHILE 还有输入?
        计算每加仑行驶的英里数
        打印结果
        读入加仑数和行驶的英里数
    END DO
STOP
```

Turbo Basic 程序

```
' This program Computes miles per gallon
INPUT NoOfGallons, MilesTravled
DO WHILE NoOfGallons < 99
    MilesPerGallon = MilesTravled / NoOfGallons
    PRINT MilesPerGallon
    INPUT NoOfGallons, MilesTravled
LOOP
PRINT "End of Job"
END
```

2.

伪代码

```
START
    读入一个销售额
    DO WHILE 还有输入?
        计算佣金
        打印结果,
        读入销售额
    END DO
    打印操作结束信息
STOP
```

Turbo Basic 程序

```
' This program Calculates Sales commissions
INPUT SalesAmount
DO WHILE SalesAmount < 99999
    Commission = .15 * SalesAmount
    PRINT Commission
```

```
    INPUT SalesAmount
LOOP
PRINT "DONE!"END
```

3.

伪代码

```
START
    读入五天的销售额
    DO WHILE 还有输入?
        计算五天销售额之和
        打印结果
        读入五天销售额
    END DO
    打印操作结束信息
STOP
```

Turbo Basic 程序

```
' This program Calculates weekly Sales
INPUT MonSales, TueSales, WedSales, ThuSales, FriSales
DO WHILE MonSales < 99999
    Total = MonSales + TueSales + WedSales + ThuSales + FriSales
    PRINT Total
    INPUT MonSales, TueSales, WedSales, ThuSales, FriSales
LOOP
PRINT "DONE!"END
```

自我测验(四)

1. 4+2／2＝4+1＝5

2. 6／（5−3）＝6／2＝3
3. (4＊5）+2＝20+2＝22
4. 3+（4＊2）−（60∧2）＝3+（4＊2）−36
   ＝3+8−36＝−25
5. 2A必须写成2＊A，即：LET　C＝2＊A+D
6. 只有变量名可以用在LET的后面。正确的是：

```
LET  G = A + E
```

7. 正确。
8. 掉了一个双引号，应为：

```
PRINT "THE ANSWER IS ", F
```

9. A ＝ (3 ＋ C) ／ (D ＋ 7)
10. s ＝ ((T − 3) ˆ2 − r) ／ 6

自我测验(五)

1. 2Radius是非法的，应为2＊Radius。
2. 赋值语句要求在等号"＝"前是变量。这条语句应是：

```
LET  Area = Length * Width.
```

3. 为了求平均值，必须用3除这三个数之和：

```
LET  Average =  (Var1 + Var2 + Var3)
```

4. Name＄不是合法的字符串变量名。下面的才是合法的：

```
FirstName$ = "Alfred"
```

5. %Pi是一个整型常量，必须赋予一个整数值。
6. ＋ −2不允许在一起运算。该语句应为：

```
Amt1 = Amt2-2 * Discount
```

7. 正确。
8. Result1 ＝ (10／5 ＋ 6／2) ＊ 3 ＝ ((2 ＋ 3) ＝ 5＊3 ＝ 15
   Result2 ＝ (10／5 ＋ 6／2) ＊ 3 ＝ 2 ＋ 3 ＊ 3 ＝ 2 ＋ 9 ＝ 11
9. 将打印如下的语法错误：错误11：被0除。
10. T.
11. T.

12. 11.5
13. 48.59999847412109（计算机近似等于48.6）
14. −1.5
15. 11
16. 262144
17. 1
18. 17
19. 5（C　MOD　G * 2）=C　MOD　6，因为乘法优先级高于MOD。
20. 1
21.

伪代码

```
START
     读入雇员名字、毛收入和税
     DO WHILE 还有输入?
         计算净收入
         打印结果
         读入雇员名字、毛收入和税
     END DO
STOP
```

Turbo Basic 程序

```
' This program calculates NetPay
INPUT EmpName$, GrossPay, Taxes
DO WHILE Taxes
 < 99999
     NetPay = GrossPay - Taxes
     PRINT EmpName$, Taxes
INPUT EmpName$, GrossPay, Taxes

LOOP
PRINT "End of Job"
END
```

22.

伪代码

```
START
    读入五天的销售额
    DO WHILE 还有输入?
        计算五天销售额的平均值
        打印结果
        读入五天销售额
    END DO
STOP
```

Turbo Basic 程序

```
' This program calculates average daily sales
INPUT MonSales, TueSales, WedSales, ThuSales, FriSales
DO WHILE MonSales < 99999
    Average = (MonSales + TueSales + WedSales + ThuSales + FriSales) / 5
    PRINT Average
    INPUT MonSales, TueSales, WedSales, ThuSales, FriSales
LOOP
END
```

23.

伪代码

```
START
    读入顾客名字，货物名称，件数，单价    DO WHILE 还有输入?
        计算结算额
        打印顾客名字，货物名称，结算额
        读入顾客名字，货物名称，件数，单价    END DO
```

```
STOP
```

Turbo Basic 程序

```
' This program calculates Balance Due
INPUT CustName$, StockName$, Share, Price
DO WHILE Price < 99999
    BalDue = Share * Price
    PRINT CustName$, StockName$, BalDue
    INPUT CustName$, StockName$, Share, Price
LOOP
END
```

24.

伪代码

```
START
    读入 A, I, Y
    DO WHILE 还有输入?
        计算投资金额
        打印结果
        读入 A, I, Y
    END DO
STOP
```

Turbo Basic 程序

```
' This program calculates the value of an investment
' at the end of any given year
INPUT Amt, Interest, Years
```

```
DO WHILE Amt < 99999
     Value = Amt * (1 + I / 100) ^Y
     PRINT Value
     INPUT INPUT Amt, Interest, Years
LOOP
END
```

自我测验(六)

1. PRINT
2. LPRINT
3. 在打印机上打印一个空行.
4. 在屏幕上显示一个空行.
5. 赋给变量Z和Y的值显示在屏幕上的同一行中. 且至少有一个空格分隔. 如Y是正数, 将有两个空格分隔Z, Y; 若Y为负数, 则仅有一个空格将它们分隔.
6. "The total is "与存贮在变量Total中的值显示在同一行中,且若Total为正数, 则二者间就有两个空格分隔; 若Total为负数, 则有一个空格分隔.
7. 变量Z、Y的值将加在一起, 并打印出. 建议不要用PRINT或LPRINT语句完成算术运算. 下面的更好:

```
Z = X + y
LPRINT  Z
```

8. T
9. T
10. F——因为字符串变量是一个接一个打印的。

自我测验(七)

1. LPRINT; PRINT
2. T.
3. T.
4. T——在INPUT指令中可以选择地打印指示 (如, INPUT"Enter Amt";AMT)。
5. F——空格与分号的作用相同.
6. T.
7. **语句** **显示结果**

语句	显示结果
10 PRINT 8 + 5	13

```
10 PRINT "8 + 5"            8 + 5(实际上，该数据由引号括住．)
```

8. (a)中，运行窗口上显示有给用户的提示．
   (b)中，注释是给程序员的，而不是给用户的。

```
INPUT "Enter Name", Name1 $
INPUT "Enter Address" , Address1 $
```

9. 空格；负号．
10. 14（在80列的屏幕上，最后一个打印区域有24个字符）．

本章自测

1. AverageSales 含有DayTimeSales加上EveningSales的一半的和。
2. UnitPrice含有Amt1加上Amt2被Qty除后的值（仅有Amt2被Qty 除）．
3. T。
4. F——大写字是保留定．
5. T.
6. CLS.
7. F——注释可以出现在含有指令行尾。
8. 逗号．
9. %.
10. $
11. LET Hours = Hours + OvertimeHoues
12. LET Feet% = X ＼ 12
    LET Reminches = X MOD 12
    注意: 写成 LET Feet% = X / 12 是错误的，因为这样的结果会四舍五入。
    因而，运算 44 / 12 后，Feet%中的值为错误的：4.
13. LET WeekEndSales = FriSales + SatSales + SunSales
14. LET Total = Total + Amt1 + Amt2 + Amt3
15. LET Amt1 = Amt1 − 47.5

调试练习

1. 由于Amt of sales 中有空格，故它是非法的数字变量．
2. 1不能加到一个字符串变量中。
3. 给用户的提示应出现在第一条INPUT语句前。
   另外，由于这里没有第二条 INPUT 语句。将出现死循环．
4. 错误有：

(1) Total应初始设为0（这本质上是没有错误的，但所有变量都应初始化。）

(2) 在循环中，第二条赋值语句应为: Total = Total + Sale。

5. 错误有:

(1) 面积公式有错误，就为Height * Width1;

(2) 第二条输入语句的变量顺序应颠倒.

6. 应为: Amt2 = Amt1 * 3

7. 应为: Product1 = 2 * Amt3

第 四 章

自我测验(一)

1. 伪代码为:

伪代码

```
START
    输入雇员姓名和应得工资
    DO WHILE 还有输入?
        IF 应得工资 < 50000 THEN
            联邦所得税为 20%
        ELSE
            联邦所得税为 25%
        END IF
        打印雇员姓名和联邦所得税
        输入雇员姓名和应得工资
    ENDDO
STOP
```

程 序

```
'This program calculates federal income tax
CLS
PRINT "Enter Name and Gross Pay"
PRINT "Type Stop, 999999 when done"
INPUT EmpName$, GrossPay
DO  WHILE GrossPay < 999999
    IF  GrossPay < 50000 THEN
        FedlTax = .20 * GrossPay
```

```
    ELSE
        FedlTax = .25 * GrossPay
    END IF
    LPRINT EmpName$; " owes "; FedlTax
    INPUT EmpName$, GrossPay
LOOP
END
```

2. 伪代码为:

伪代码

```
START
    初始化奖金总额
    输入雇员姓名和工资
    DO WHILE 工资 > 0?
        IF 工资 <= 25000 THEN
            奖金为 10%
        ELSE
            奖金为 8%
        END IF
        打印雇员姓名和奖金
        将奖金加入奖金总额
        输入雇员姓名和工资
    ENDDO
    打印奖金总额
STOP
```

程 序

```
'This program prints bonuses and calculates total bonuses paid
CLS
TotalBonus = 0
PRINT "Enter Name and Salary"
PRINT "Type END, 0 When Done"
INPUT EmpName$, Salary
DO  WHILE Salary > 0
    IF  Salary <= 25000 THEN
        Bonus = .10 * Salary
    ELSE
        Bonus = .08 * Salary
    END IF
```

```
    LPRINT "Bonus for "; EmpName$; " is "; Bonus
    TotalBonus = TotalBonus + Bonus
    INPUT EmpName$, Salary
LOOP
LPRINT "Total Bonus is "; Totalbonus
END
```

自我测验(二)

1. 第一行少了个"THEN"
2. Amt 是一个数字变量，应与127相比较而不是-127"。
3. 每个IF块都应有END IF
4.
```
LET PrintSmall = A
IF  B < PrintSmall THEN
    PrintSmall = B
END IF
IF  C < PrintSmall THEN
    PrintSmall = C
END IF
```
5. T
6. "Adams" < "Baker"(A < B)
7. IBM < "ibm" (大写小于小写)。
8. "Turbo" < "Turbo Basic" (短串小于长串)。
9. 123等于0123(数字变量比较按代数规则)
10. 该比较无效，因为字符串变量不能与数字变量比较，这样做引起系统错误:

    ERROR 418: numeric experession requires relational operator.

11. " "小于"b̸"(b̸为空格)(" "为空串)
12. **Pseudocode**

```
START
   Enter Name, Hours and Code
   DO WHILE there is still input
      Initialize Rate
      IF Code = 1 THEN
         Rate = 5.00
      ENDIF
      IF Code = 2 THEN
         Rate = 7.00
      ENDIF
```

```
      IF Code = 3 THEN
         Rate = 10.00
      ENDIF
      Calculate Wages
      Print Wages
      Enter Name, Hours and Code
   ENDDO
STOP
```

Turbo Basic Program

```
'This program computes wages for employees
CLS
PRINT "Enter Name, Hours and Code"
PRINT "Type End, 99, 9 when done"
INPUT EmpName$, Hours, Code
DO  WHILE Code <> 9
    Rate = 0
IF  Code = 1 THEN
    Rate = 5.00
END IF
IF  Code = 2 THEN
    Rate = 7.00
END IF
IF  Code = 3 THEN
    Rate = 10.00
END IF
Wages = Hours * Rate
PRINT EmpName$; " earned "; Wages
INPUT EmpName$, Hours, Code
LOOP
END
```

自我测验(三)

注意: 第二至五题的解答仅供参考.

1. a. 10; b. 40; c. 10
2.

```
CLS
PRINT "Enter Number of Ounces; 9999 WHEN Done"
INPUT Ounces


DO  WHILE Ounces < 9999
    IF  Ounces > 1 THEN
        Postage = .25 + (Ounces - 1) * .17
        PRINT "Postage is "; Postage
    ELSE
        IF  Ounces > 0 THEN
            Postage = .25
            PRINT "Postage is "; Postage
        ELSE
            PRINT "Ounces entered as a negative number or zero "
        END IF
    END IF
    INPUT Ounces
LOOP
PRINT "End of Program"
END
```

```
3.  CLS
    PRINT "Enter Code (R, G, or B) and Price"
    PRINT "Enter X, 0 to Exit"
    INPUT Code$, Price
    DO  WHILE Code$ <> "X"
        IF  Code$ = "R" THEN
            Discount = .30
        ELSEIF Code$ = "G" THEN
            Discount = .25
        ELSEIF Code$ = "B" THEN
            Discount = .20
        ELSE
            PRINT "No discount"
            Discount = 0
        END IF
        SellingPrice = Price - Price * Discount
        PRINT "Selling Price is "; SellingPrice
        INPUT Code$, Price
    LOOP
    PRINT "End of Program"
    END
```

```
4.  CLS
    PRINT "Enter Student Name and Number of Credits"
    PRINT "Enter STOP, 99 when done"
    INPUT StudName$, NoOfCredits
    DO  WHILE NoOfCredits < 99
        IF  NoOfCredits <= 12 THEN
            Tuition = NoOfCredits * 200
            ELSEIF NoOfCredits < 16 THEN
                Tuition = 2600
                ELSEIF NoOfCredits < 19 THEN
                    Tuition = 3200
                    ELSE
                        Tuition = 0
                        PRINT "Error"
        END IF
        PRINT "Tuition for "; StudName$; " is "; Tuition
        INPUT StudName$, NoOfCredits
    LOOP
    END
```

```
5.  CLS
    PRINT "Enter Salary and Sales Amount"

    PRINT "Enter 999999, 0 when done"
    INPUT Salary, Sale
    DO  WHILE Salary < 999999
        IF  Salary > 1000 THEN
            IF  Sale > 5000 THEN
                Commission = Sale * .08
            ELSE
                Commission = Sale * .05
            END IF
        ELSE
            Commission = Sale * .065
        END IF
        PRINT Commission
        INPUT Salary, Sale
    LOOP
    END
```

下面的判段表可用于程序的准备:

Condition 1	Condition 2	Action
Salary > 1000	Sale > 5000	Commission = Sale * .08
Salary > 1000	Sale <= 5000	Commission = Sale * .05
Salary <= 1000	Sale = anything	Commission = Sale * .065

自我测验(四)

1. 不应使用隐含操作数.
2. 不应使用隐含操作数.
3. IF只能使用一次.
4.
```
IF Amt1 >= 100 AND Amt1 <= 1000 THEN
    PRINT "A-Ok"
END IF
```
5.
```
IF Amt1 > 100 AND Amt1 < 1000 THEN
    PRINT "A-Ok"
END IF
```
6. 不能. 没有包括两个端点.
7. 不能. 扩大了Total的范围.
8. 能.
9. (a) 用复合条件句
```
IF  Age < 21 THEN
    PRINT "Does Not Meet Criteria"
ELSEIF Income > 40000 OR Credit >= 5000 THEN
    PRINT "Meets Criteria"
ELSE
    PRINT "Does Not Meet Criteria"
END IF
```
   (b) 用条件句嵌套
```
IF  Age >= 21 AND (Income > 40000 OR Credit >= 5000) THEN
    PRINT "Meets Criteria"
ELSE
    PRINT "Does Not Meet Criteria"
END IF
```
10.
```
IF  YearEndBonus > 2500 AND OutstandingBills < 2000 THEN
    PRINT "Go On Vacation"
ELSE
    PRINT "Stay Home This Year"
END IF
```


19A8-32

11.

伪代码

```
START
    初始化 FemCtr, MaleCtr, TotEmp
    输入性别和工资
    DO WHILE 还有输入?
        将 TotEmp 加 1
        IF 为女性且工资在 50000~1000000 之间 THEN
            将 FemCtr 加 1
        END IF
        IF 为男性且工资在 50000~1000000 之间 THEN
            将 MaleCtr 加 1
        END IF
        输入性别和工资
    ENDDO
    计算工资在 $ 50000~ $ 1000000 之间的女性雇员所占的百分比
    计算工资在 $ 50000~ $ 1000000 之间的男性雇员所占的百分比
    打印结果
STOP
```

程 序

```
'This program prints % of Males and Females who Earn Between
'$50,000 and $100,000
CLS
FemCtr = 0
MaleCtr = 0
TotEmp = 0
PRINT "Enter Sex of Employee (M/F) and Salary"
PRINT "Type Q, 999999 when done"
INPUT Sex$, Salary
DO WHILE Salary < 999999
    LET TotEmp = TotEmp + 1
    IF  Sex$ = "F" AND Salary >= 50000_
            AND Salary <= 100000 THEN
        FemCtr = FemCtr + 1
    END IF
    IF  Sex$ = "M" AND Salary >= 50000_
            AND Salary <= 100000 THEN
        MaleCtr = MaleCtr + 1
    END IF
    INPUT Sex$, Salary
```

```
LOOP
PercentFemales = (FemCtr / TotEmp) * 100
PercentMales = (MaleCtr / TotEmp) * 100
PRINT "% of Females who Earn Between 50000 and 100000 is "; _
          PercentFemales
PRINT "% of Males who Earn Between 50000 and 100000 is "; _
          PercentMales
END
```

12.

伪代码

```
START
  输入货名，单价，库存数量    DO WHILE 还有输入?
    初始化折扣率
    IF 单价在 5.00~10.00 之间 AND 库存数量大于 100 THEN
      折扣率为 2%
    END IF
    IF 单价在 5.00~10.00 之间 AND 库存数量不大于 100 THEN
      折扣率为 1%
    END IF
    IF 单价大于 10.00 OR 小于 5.00 THEN
      折扣率为 3%
    END IF
    打印折扣率
    输入货名，单价，库存数量
  ENDDO
STOP
```

程　序

```
'This program calculates discounts for specified items
CLS
PRINT "Enter Part Description, Unit Price, Quantity on Hand"
PRINT "Type End, 0, 0 When Done"
INPUT Part$, Price, Qty
DO WHILE Price <> 0
    Discount = 0
    IF  Price >= 5.00 AND Price <= 10.00_
            AND Qty > 100 THEN
        Discount = .02 * Price
    END IF
```

```
    IF   Price >= 5.00 AND Price <= 10.00_
                 AND Qty <= 100 THEN
        Discount = .01 * Price
    END  IF
    IF   (Price > 10.00 OR Price < 5.00)_
                 AND Qty >= 250 THEN
        Discount = .03 * Price
    END IF
    PRINT "Discount is "; Discount; " for "; Part$
    INPUT Part$, Price, Qty
LOOP
END
```

本章自测

1. IF在语句中只能使用一次.

```
IF  A = B  OR  A = C  THEN
    PRINT "A OK"
END  IF
```

2. 正确. 但是取好把打印(PRINT)语句与算术运算分开.

```
IF  Code$  =  "Delete"  THEN
    Ctr  =  Ctr + 1
    PRINT  Ctr
END  IF
```

3. 正确. 条件中的每个部分可以是变量、数值或者是算术表达式. A+B 是一个算术表达式, 因而是允许的; 也可以写成:

```
LET  D  =  B  +  A
IF  C  <  A+B  THEN
    PRINT "C OK"
END  IF
```

4. 应该加上括号才符合逻辑.

```
IF  (A < 21  OR  A = 21)  AND  (A = 5  OR  A > 5)  THEN
    PRINT "A OK"
END  IF
```

若没有括号，IF 块被认作:

```
IF  A < 21  OR  (A = 21  AND  A = 5)  OR  A > 5  THEN
    PRINT "A OK"
END  IF
```

因为 A 不能同时等于 21 和 5，因而变成:

```
IF  A < 21  OR  A > 5  THEN
    PRINT "A OK"
END  IF
```

若把程序写成如下形式，则更完美:

```
IF  A <= 21  and  A >= 5  THEN
    PRINT "A OK"
END  IF
```

5. 应改写为:

```
IF  A <> 3. AND  A <> 4  THEN
    PRINT "A IS NOT 3 OR 4"
END  IF
```

6. AND；OR；NOT.
7. 只有A是整数时，才相同；若为小数，结果不同.

注意：下面的答案并不是唯一的：

8.
```
IF  A <= 13  AND  A >= 3  THEN
        PRINT  "A OK"
    END  IF
```

9.
```
IF  A > 3  AND  A < 13  THEN
        PRINT  "A OK"
    END  IF
```

10.
```
IF  A = B  AND  C = D  AND  E = F  THEN
        PRINT  "A OK"
    ELSE
        PRINT  "A NOT OK"
    END  IF
```

11.
```
IF  A >= 0  THEN
        PRINT  "Absolute Value of "; A; " is "; A
    ELSE
```

```
        PRINT  "Absolute Value of "; A; " is "; -1 * A
    END  IF
```

12.
```
IF  Code$  >= "9"  OR  Code$  < "0"  THEN
        PRINT  Code$ ; "IS NOT NUMERIC"
    ELSE
        PRINT  Code$ ; "IS NUMERIC"
    END  IF
```

13. 打印"Equal".

14.
```
IF  Weight >= 120  AND  Age >= "25  AND  Age <= 30  THEN
        PRINT  "I am interseted"
    END  IF
```

注：由于条件全部用 AND 连接，故不必加括号.

15. 伪代码为:

```
START
   Input G, T
   DO WHILE there is more data
      IF G = 0 THEN
         Next Statement
      ELSE
         Let M = T/G
         Print "NO OF MILES PER GALLON IS ", M
      ENDII
      Input G, T
ENDDO
STOP
```

程序为:

```
INPUT G, T
DO  WHILE G <> 99
     IF G = 0 THEN
        EXIT IF
     ELSE
        LET M = T / G
        PRINT "NO OF MILES PER GALLON IS "; M
     END IF
     INPUT G, T
LOOP
END
```

16. 程序为:

(a)
```
IF  A > 100.6 OR A < 98.2 THEN
        PRINT "A NOT OK"
    ELSE
        PRINT "A OK"
    END IF
```

或:

```
(b) IF  A <= 100.6 AND A >= 98.2 THEN
        PRINT "A OK"
    ELSE
        PRINT "A NOT OK"
    END IF
```

17. 错误。当A<92.8时，将什么也不打印，因而必须写成:

```
IF  A <= 100.6 THEN
    IF  A >= 98.2 THEN
        PRINT "A OK"
    ELSE
        PRINT "A NOT OK"
    END IF
ELSE
    PRINT "A NOT OK"
END IF
```

18.

伪代码

```
START
    设最大值变量 PrintLarge 为 0
    IF  A  >  B  THEN
        将 A 移入 PrintLarge
    ELSE
        将 B 移入 PrintLarge
    ENDIF
    IF  C  >  PrintLarge  THEN
        将 C 移入 PrintLarge
    ENDIF
STOP
```

程　序

```
LET  PrintLarge  =  0
IF  A  >  B  THEN
    PrintLarge  =  A
ELSE
    PrintLarge  =  B
ENDIF
IF  C  >  PrintLarge  THEN
```

```
    PrintLarge = C
  ENDIF
```

19.

伪代码

```
START
    输入销售额
    DO WHILE 还有输入?
        IF 销售额 > 500 THEN
            设定佣金比率为 5%
        ELSEIF 销售额 >= 200 AND 销售额 <= 500 THEN
            设定佣金比率为 3%
        ELSE
            设定佣金比率为 2%
        ENDIF
        佣金 = 销售额 * 佣金比率
        打印佣金
        输入销售额
    ENDDO
STOP
```

程 序

```
PRINT "Enter Sales Amount: Type 99999 When Done"
INPUT SalesAmount
DO  WHILE SalesAmount < 99999
    IF  SalesAmount > 500 THEN
        Discount = .05
    ELSEIF SalesAmount >= 200 AND SalesAmount <= 500 THEN
        Discount = .03
    ELSE
        Discount = .02
    END IF
    DiscountAmount = SalesAmount * Discount
    PRINT DiscountAmount
    INPUT SalesAmount
LOOP
END
```

20.

伪代码

```
START
    输入每小时的报酬和工作时间
    DO WHILE 工作时间 <> 99
        IF 工作时间 <= 40 THEN
            工资 = 每小时的报酬 * 工作时间
        ELSE
            工资 = 40*每小时的报酬 + _
                  (工作时间 40 ) * (每小时的报酬*1.5)
        ENDIF
        打印工资
        输入每小时的报酬和工作时间
    ENDDO
STOP
```

程 序

```
PRINT "Enter Rate and Hours — Type 99, 99 when Done"
INPUT HourlyRate, HoursWorked
DO WHILE HoursWorked <> 99
    IF HoursWorked <= 40 THEN
        Wages = HourlyRate * HoursWorked
    ELSE
        Wages = 40*HourlyRate + _
              ( HoursWorked 40 ) * (HourlyRate*1.5)
    ENDIF
    PRINT Wages
    INPUT HourlyRate, HoursWorked
LOOP
END
```

第 五 章

自我测验(一)

1. 6次
2. 5次
3. 0次
4. 死循环，在循环中Ctr不改变
5. 6次
6.

```
' this program calculates a class average
   LET Total = 0
   LET ExamCtr = 0
   LET MoreInput$ = "Y"
   DO WHILE MoreInput$ = "Y" OR MoreInput$ = "y"
        PRINT "Enter EXAM"
        INPUT Exam
        LET ExamCtr= ExamCtr+ 1
        LET Total = Total + 1
        PRINT "More Input (Y/N)?"
        INPUT MoreInput$
   LOOP
   LET Average = Total / ExamCtr
   PRINT "The Class Average is:"; Average
   END
```

7. 对于（a），可以输入和累加任意个"price"；如果999是在第20，26或500时输入，将终止循环，并打印总和，程序依赖于用户精确输入 25 个"Price"。
   对于（b）循环执行固定次数，即25次。因此，只有25个"price"可被累加。
8.

```
' this program calculates a class average
LET Sum = 0
LET EvenNo = 2
DO WHILE EvenNo <= 2000
     LET Sum = Sum + EvenNo
     LET EvenNo= EvenNo+ 2
```

```
LOOP
PRINT "The sum of even numbers from 2-2000 is:"; Sum
END
```

9. F——循环根本不执行
10. T.

自我测验(二)

1. DO WHILE MoreInput $ = "Y" OR MoreInput $ = "y"

   或:

   DO WHILE MoreInput $ <> "N" AND MoreInput $ <> "n"

   对于任何不是"N"或"n"的MoreInput $ 的值，程序都应执行循环（在后一种情况下要用"AND"，以保证不是"N"或"n"时能执行循环).
2. 死或无限
3. WEND
4. T
5. 迭代
6. EXIT
7. T
8. DO WHILE … … LOOP; DO UNTIL … LOOP; WHILE … WEND

   执行完一次循环之后，也可以用DO … LOOP WHILE …… 或者DO … LOOP UNTIL …… 来测试条件.
9.

```
'This program uses a DO WHILE ... LOOP
CLS
LPRINT "                Wage Report for 20 Hours Per Week"
LPRINT
LPRINT "                Hourly Pay", "Weekly Pay"
HourlyRate = 4.00
Increment = .25
HoursWorked = 20
DO  WHILE HourlyRate <= 6.00
    WeeklyPay = HoursWorked * HourlyRate
    LPRINT "  ", HourlyRate, WeeklyPay
    HourlyRate = HourlyRate + Increment
LOOP
LPRINT "End of Report"
END

'This program uses a DO UNTIL ... LOOP
CLS
LPRINT "                Wage Report for 20 Hours Per Week"
LPRINT
LPRINT "                Hourly Pay", "Weekly Pay"
LPRINT
HourlyRate = 4.00
Increment = .25
HoursWorked = 20
```

```
DO  UNTIL HourlyRate > 6.00
    WeeklyPay = HoursWorked * HourlyRate
    LPRINT " ", HourlyRate, WeeklyPay
    HourlyRate = HourlyRate + Increment
LOOP
LPRINT "End of Report"
END
```

注意: 程序开始处LPRINT语句中的"　　"(空格)和逗号是为了输出结果的对齐而加入的.

10.
```
CLS
Total = 0
Number = 1
WHILE Number < 100
    Total = Total + Number
    Number = Number + 2
WEND
LPRINT Total
END
```

自我测验(三)

1. NEXT语句应为NEXT　N

2.
```
CLS
FOR N = 1 TO 5
    LPRINT N, N * 2, N * 3, N * 4, N * 5
    LPRINT
NEXT N
END
```

3.
```
CLS
N = 1
DO WHILE N <= 5
    LPRINT N, N * 2, N * 3, N * 4, N * 5
    LPRINT
    N = N + 1
LOOP
END
```

4. Yes, the loops are correct.

5.
```
CLS
Total = 0
Ctr = 1
DO  UNTIL Ctr > 10
    INPUT Amt
    Total = Total + Amt
    Ctr = Ctr + 1
LOOP
PRINT Total
END
```

6.
```
CLS
Total = 0
FOR Ctr = 1 TO 10
    INPUT Amt
    Total = Total + Amt
NEXT Ctr
PRINT Total
END
```

7.
```
CLS
Total = 0
FOR Number = 2 TO 2000 STEP 2
    Total = Total + Number
NEXT Number
PRINT Total
END
```

8.
```
CLS
Total = 0
INPUT N%
INPUT M%
FOR Number = N% TO M%
    Total = Total + Number
    PRINT Total
NEXT Number
END
```

注：可分别用 N%，M%代替 N 和 M，隐含定义变量为整数。

9.
```
CLS
Total = 1
INPUT N
FOR Ctr = N TO 1 STEP -1
    Total = Total * Ctr
NEXT Ctr
PRINT "N FACTORIAL = "; Total
END
```

10.

a.	b.	c.	d.
1	.5	10	2
2	1	5	4
3	1.5	0	6
4	2		8
5	2.5		10
6	3		12
7	3.5		
8	4		
9	4.5		
10	5		

10. 程序将打印如下：

e. 无打印，因为2／3（Y／Z）小于X的初始值：1.

自我测验(四)

1.
```
CLS
SUM = 0
EvenNum = 2
DO  UNTIL EvenNum > 2000
    GOSUB AddEven
LOOP
PRINT Sum
END
```

2.
```
CLS
Sum = 0
EvenNum = 50
DO  UNTIL EvenNum > 3002
    GOSUB Addeven
LOOP
PRINT Sum
END
```
3.
```
CLS
Sum = 0
Number = 1
DO  WHILE Number < 2499
    GOSUB AddNumber
LOOP
PRINT Sum
END
AddNumber:
    Number = Number + 1
    Sum = Sum + Number
RETURN
```
4.
```
CLS
Result = 1
INPUT Number
FOR Factor = Number TO 1 STEP -1
    GOSUB CalculateFactorial
NEXT Factor
PRINT Number, "Factorial = "; Result
END
CalculateFactorial:
    Result = Result * Factor
RETURN
```
5. xyz前必须有END语句，xyz后必须跟冒号。
6. 不能用Error作为标号，因为它是Turbo Basic的保留字。
7. EnterData 应为EnterData:
   END SUB 应为 RETURN。
8. EnterData; 应为EnterData:
9. 在一个GOSUB标号内部不能有GOSUB 同一标号，一个子程序只能调用其它子程序，而不能调用它自己。
10. END 应在Print Routine 之前。
11. 当一个指令序列在程序中多处应用时，应用子程序最有利。在一定条件下需要一个联系紧密的指令集时，应用子程序也有好处。例如:
```
IF Amt1 <0 THEN GOSUB NegativeBalance
```
12. 能。一个子程序可以调用另一个子程序（这叫做嵌套）。

本章自测

1. F——当一段程序反复执行时，要用循环编程。
2. F——FOR语句中的计数器或变量必须是数值，但不必是整数（但是，建议用户使用整型变量）。

3. T
4. F———一个子程序可在程序中多处引用.
5. 需要有STEP分句，指出一负的增量.
6. STEP 分句必须指出一个正的增量.
7. 计数器变量必须是数值.
8. 如果循环计数器没有修改循环将无限重复（一个死循环）. 因为循环条件永远不能满足.
9. 只要现存条件连续满足，就执行DO WHILE … LOOP; DO UNTIL … LOOP 一直执行, 直到某条件满足.
10. 自顶而下模块化方法使得程序的调试和修改更容易，此外，用它可得到较好的程序设计.

第 六 章

自我测验(一)

1. a. #1后要有逗号.
   b. Name $ 是个保留字，用Employee Name或某种变型.
2. 需用EOF(filenum).
3. 文件名应在引号内.
5. T
6. F——从顺序文件读入时方需检查文件的结束符.
7. T
8. 顺序；随机；二进制.
9. 头；不需要.
10. 记录
11. WRITE #
12. INPUT #
13. 当需要处理一项事务时，交互处理一般通过键盘或终端完成. 而在批处理方式下，与一文件有关的所有事务收集在一起，一块处理.
14. 存取第47号记录，必须读过前46条记录.
15. 文件是相关记录的集合，记录是相关字段的集合.

自我测验(二)

1. 两个美元符.
2. 星号填充主要用于保护支票上的数目，以防未受权的改动.

3. LPRINT USING "####"; 457
   LPRINT USING "####"; 38
   LPRINT USING "####"; 5
4. INPUT AnyField
   PRINT USING "####.####"; AnyField
5. 5.43−
6. 550.50
7. 550.50
8. %145.432(长度错误)
9. 145.43
10. * * * * 375.85

自我测验(三)

1. 格式串与变量之间不得用逗号，应用分号.
2. ！只能用于串变量.
3. LPRINT 后要带USING 一词.
4. 左.
5. !
6. &
7. 5
8. F——＼、!与&只能用于打印或显示串变量的格式串中.
9. T
10. F——每个反斜杠算一个字符位置，所以输出字符数等于空格数加二.

B208-32 ②

自我测验(四)

1. 行计数器.
2.
```
LPRINT ...
LineCtr = LineCtr + 1
IF LineCtr >= 30 THEN
    GOSUB PrintHeadings
END IF
```
3. 打印格式.
4.
```
LPRINT TAB(33); "INVENTORY REPORT"
LPRINT
LPRINT
LPRINT TAB(7); "ITEM NUMBER", TAB(28); "DESCRIPTION";
LPRINT TAB(49); "ON HAND", TAB(67); "COST"
```

5. F——TAB函数不能与LPRINT USING 或PRINT USING共用.
6. F——如果光标超过了TAB函数的位置，则打印机往前进一行，然后在指定位置上打印.
7. WIDTH
8.
```
WIDTH "LPT1:", 120
LPRINT TAB(52); "INVENTORY REPORT"
LPRINT
LPRINT
LPRINT TAB(22); "ITEM NUMBER"; TAB(46); "DESCRIPTION";
LPRINT TAB(70); "ON HAND"; TAB(90); "COST"
```
9.
```
PRINT TAB(7); "EMPLOYEE"; TAB(30); "HOME"; TAB(45); "HOME"
PRINT TAB(9); "NAME"; TAB(28); "ADDRESS"; TAB(45); "PHONE"
```
10.
```
LPRINT CHR$(12)
```

本章自测

1. 0.05
2. 1.05
3. $ 26.54
4. 2
5. 4
6. $0.00
7. $25.00
8. 8
9. 123
10. 2
11. 表头，栏标题.
12. 打印格式表
13. LPRINT CHR$(12)
14. LET S$ =DATE$
    LPRINT S$
    或 LPRINT DATE$
15. WIDTH
16. 明细
17. 例外
18. 一览
19. PRINT TAB(32);"INVENTORY REPORT"
20. PRINT USING "$##,###.##"; 37750.25

21. T
22. T
23. T
24. F——在编辑字母前将会显示一个%，表明产生了溢出。
25. F——如果已过了一TAB中指定的位置，则会到达下一行中的那个位置。

第 七 章

自我测验

1. 控制中断处理
2. 控制字段
3. 控制字段
4. 保持域
5. 当前输入记录的控制字段到保持域
6. 增加一个控制总计并打印，当细目打印需要时
7. 完成控制中断过程
8. 打印最后记录，增加最后的值到控制总计，人工干预控制中断，打印最后总计。
9. 控制中断
10. 控制中断
11. 控制总计；控制总计；输入控制字段到保持域
12. 细目打印
13.

```
LET LineCt = 0
LET AreThereMoreRecords$ = "Yes"
OPEN "Warehs.Dat" FOR INPUT AS #1
GOSUB ReadModule
GOSUB PrintHeadings
WHHold = Warehouse
DO  WHILE AreThereMoreRecords$ = "Yes"
    IF  WHHold <> Warehouse THEN
        GOSUB ControlBreak
    END IF
    GOSUB DetailProcessing
    GOSUB ReadModule
LOOP
GOSUB ControlBreak
CLOSE #1
END
```

14.

```
DetailProcessing:
    IF  LineCt > 25 THEN
        GOSUB PrintHeadings
    END IF
    LPRINT TAB(22);
    LPRINT PartNo;
    LPRINT TAB(43);
    LPRINT QtyOnHand
    PartTotal = PartTotal + 1
    LineCt = LineCt + 1
RETURN
```

15.

```
ControlModule:
    LPRINT TAB(30); " TOTAL NUMBER OF ITEMS STORED IN WAREHOUSE
    LPRINT TAB(74);
    LPRINT USING "#,###"; PartTotal
    WHHold = Warehouse
    PartTotal = 0
    IF  AreThereMoreRecords$ = "Yes" THEN
        GOSUB PrintHeadings
    END IF
RETURN
```

本章自测

1. 细目
2. 小结
3. F——通常，打印小结表比打印细目表快
4. 控制中断或小结
5. 控制
6. 控制中断
7. 库存数
8.

```
CLS
GOSUB InitializeModule
GOSUB ReadModule
Let WHHold = WHNo
GOSUB PrintHeadings
DO  WHILE AreThereMoreReqords$ = "Yes"
        IF  WHHold <> WHNo THEN
                GOSUB ControlBreak
        END IF
        GOSUB DetailProcessing
        GOSUB ReadModule
LOOP
GOSUB ControlBreak
GOSUB EndOfJob
```

9.

```
DetailProcessing:
  IF  LineCounter > 25 AND AreThereMoreRecords$ = "Yes" THEN
       GOSUB PrintHeadings
  END IF
  LET TotalItems = TotalItems + 1
  LET TotalValueOfInv = TotalValueOfInv + QtyOnHand * Unit Price
RETURN
```

10.

```
ControlBreak:
        LPRINT TAB(16);
        LPRINT USING "**"; WHHold;
        LPRINT TAB(33);
        LPRINT USING "*,***"; TotalItems;
        LPRINT TAB(59);
        LPRINT USING "$***,***,***.**"; TotalValueOfInv
        LET TotalItems = 0
        LET TotalValueOfInv = 0

        IF NOT EOF(1) THEN
                WHHold = WHNo
        END IF
RETURN
```

11. 当控制中断发生和最后一个记录处理后，执行ControlBreak；当最后一个记录处理后，不希望将一个不存在的库存数移入保持域。
12. 在end-of-job模块，可简单重复ControlBreak中的打印指令。
13. 将总计加在ControlBreak，比加在DetailProcessing模块中好；这将更有效，因需要更少的附加值。
14. 一个次级控制中断
15. F——排序是依主字段中的辅字段排定的。

第 八 章

自我测验(一)

1. 语法错误。
2. F——运行时错误在运行过程中检查

3. F——编译器检查语法错误，而不是逻辑错误
4. T——确信证实的子程序工作正常.
5. TRON；TROFF
6. T
7. 先在Trace窗口中显示[20]，要继续运行时，必须按Alt-F9. ?提示符出现时，若输入给定的输入数据，那么 Trace 窗口显示.

    [20][30][40]
    [70][80][90]
    [100][110][40]
    [50][60][90][100]
    [110][40][50][60]
    [90][100]
    [110][999]

   根据 Trace 窗口大小的不同，上述信息分段时可能有所差别.
8. 由跟踪信息可知；FOR 语句的起始值大于终止值，因此该循环从不执行.
9. PRINT
10. F——Ctrl-Break 组合键用来中断死循环的执行.

自我测验(二)

1. 因为输入数据时，数据录入员可能出错．错误控制过程验证数据以减少不正确输出的可能.
2. 舍入—但是，如果X为偶数，则X·5往低舍入（为X）；而X为奇数时．X·5往高舍入（为 X+1）.
3. F–即使最好的程序也不能查出所有的输入错误；不过，好的程序将查出许多输入错误.
4. T
5. T
6. 浮点数.
7. 当数据精度要求在六位有效数字之上时，数字变量应定义为双精度型.
8. T
9. T
10. 测试字符串字段应包含一个大于“空格”的值.

自我测验(三)

1. SELECT CASE语句应用于一个字段或变量包含编码数据，并根据输入的编码决

定要执行相应子程序时.

2. 控制表; 跟踪记录

3.

```
SELECT CASE MaritalStatus
     CASE = 1
          GOSUB ProcessSingle
     CASE = 2
          GOSUB ProcessMarried
     CASE = 3
          GOSUB ProcessSeparated
     CASE = 4
          GOSUB ProcessDivorced
     CASE = 5
          GOSUB ProcessWidowed
     CASE ELSE
          GOSUB ErrorRoutine
END SELECT
```

4. 输入确认

5.

```
Verify$ = "N"
DO  UNTIL Verify$ = "Y"
     GOSUB GetDataRoutine
LOOP
END
GetDataRoutine:
     PRINT "Enter Salesperson name: ";
     INPUT Salesperson$
     PRINT "Enter amount of sale: ";
     INPUT SalesAmt
     PRINT
     PRINT "Salesperson Name is: "; Salesperson$
     PRINT "Amount of sale is: "; SalesAmt
     PRINT "Is this correct (Y or N) ?"
     INPUT Verify$
RETURN
```

6. F—在某些情况下，将错误信息输入简报或控制表，并继续处理而不是中止运行，可能更好些.

7. 开关或标志

8.

```
ValidateDataRoutine:
     ErrorFlag = 0
     IF  SocSecNo < 0 THEN
          PRINT "Invalid Social Security number"
          ErrorFlag = 1
     END IF
     IF  EmployeeName$ <= " " THEN
          PRINT "Missing employee name"
          ErrorFlag = 1
     END IF
     IF  PayRate < 4.00 OR PayRate > 25.00 THEN
          PRINT "Invalid pay rate"
```

```
            ErrorFlag = 1
        END IF
        IF  ErrorFlag >= 1 THEN
            ErrorCount = ErrorCount + 1
        END IF
    RETURN
```

9. ValidateDataRoutine:

```
        ErrorFlag = 0
        IF  SocSecNo < 0 THEN
            PRINT "Invalid Social Security number"
            ErrorFlag = 1
        END IF
        IF  EmployeeName$ <= " " THEN
            PRINT "Missing employee name"
            ErrorFlag = 1
        END IF
        IF  PayRate < 4.00 OR PayRate > 25.00 THEN
            PRINT "Invalid pay rate"
            ErrorFlag = 1
        END IF

        IF  ErrorFlag >= 1 THEN
            ErrorCount = ErrorCount + 1
        ELSE
            CorrectCount = CorrectCount + 1
        END IF
        TotalRecords = TotalRecords + 1
    RETURN
```

10. ON KEY

本章自测

1. F–只能寄希望于减少输入错误，而不能将它们完全清除.
2. 控制表或跟踪记录
3. 确认（验证）
4. 整型
5. 5.5E+1
6. 双精度数
7. 分类
8. 功能键F4被按
9. DO WHILE 语句必须为:

```
DO  WHILE   Verify$ < >"N" AND Verify$ < > "n"
```

原来 DO WHILE 语句中的 OR 意即循环一直执行.

10.

```
Verify$ = "Y"
DO  WHILE Verify$ = "Y"

    GOSUB EnterAndEchoData
LOOP
SELECT CASE StructureCode
    CASE = 1
        GOSUB ResidentialRoutine
    CASE = 2
        GOSUB CommercialRoutine
    CASE = 3
        GOSUB VacantLotRoutine
    CASE ELSE
        GOSUB InvalidStructure
END SELECT
END
EnterAndEchoData:
    PRINT "Enter type of structure ";
    INPUT StructureCode
    PRINT "Enter appraised value ";
    INPUT Value
    PRINT "The structure type is: "; StructureCode
    PRINT "The value is: "; Value
    PRINT "Is this correct? (Y or N) ";
    INPUT Verify$
RETURN
```

第 九 章

自我测验(一)

1. F1.
2. F —— Help功能是与上下文有关的，具体显示的信息随当时要执行命令的不同而不同.
3. FL（即 File 中的首字母 F，Load 中的 L).
4. 下拉选单.
5. 上托选单.
6. F —— 许多，但不是所有的，下拉选单有上托选单.
7. Esc（或 Escape).
8. 当前.
9. Tile; Window.

10. F —— 只有 Run 和 Edit 窗口均缩放。
11. T —— 使之成为 TBCONFIG · TB 文件的一部分。
12. T
13. Window; Close
14. T
15. F —— 不用进入相应的命令，Goto 选项可返回到当前窗口。

自我测验(二)

1. F——取决于当时屏幕上显示的内容。
2. T
3. 重新设置窗口的大小和位置
4. F2
5. TBCONFIG.TB（或其他扩展名为.TB 的配置文件）
6. F——是 Setup 命令下的 Color 选项
7. F——程序在运行之前将被保存。即，如果编写程序时未运行该程序就关掉计算机，Auto save edit 选项将不会保存该程序。
8. 将被提示输入文件:

Load File Name
*** .BAS**

如果按Enter键，则所有的Turbo Basic文件将被显示；可用光标键加亮想要装入的程序，然后按 Enter 键选用。

9. T
10. T——先选取Options命令下的Compile to选项，然后再从上托选单中选 EXE。这样，程序文件将以编译后的形式保存。例如，如果源程序为 INVTY.BAS，那么它将以可执行文件形式存作 INVTY.EXE。为运行该文件，回到 DOS 状态下，并键入 INVTY 即可。

本章自测

1. F——只有 Edit 和 Run 窗口可放缩。
2. F——简单地选用 Window 命令下的 Open 选项即可。
3. T
4. 先一起按Scroll键和Shift键，然后按光标键可改变当前窗口大小。
5. 先按Scroll键（不按Shift键）然后按光标键可改变当前窗口位置。
6. 选用 Window 命令下的 Tile 选项。

7. 选用 Window 命令下的 Close 选项关闭当前窗口。
8. 它是一个可执行文件。为运行一个编译成×××.EXE的程序，在DOS提示符下键入×××即可。
9. 这是一个（Turbo Basic 的）配置文件。
10. 这是一个通配符，表示任何扩展名。
11. 这是一个备份文件，如果Setup命令下Miscellaneous选项中的Auto Save edit被设为 ON，并运行名字为×××的程序，那么×××.BAK 将自动存作备份文件。
12.
   a. 使 Edit 窗口成为当前窗口。
   b. 选用 Window 命令下的 Colse 选项。
   c. 选用 Window 命令下的 Tile 选项。
13. 关闭Edit和Trace两窗口。选用Winclow命令下的Stack选项。
14. 按功能键 F2。
15. 键入 Alt–X 或选用 File 命令下的 Quit 选项。

第 十 章

自我测验(一)

1. 经适当定义的数组，能占据与单独定义的变量同样大小的储空间。
2. 不行，定义数组要么全是字符串量，要么全是数字量。
3.
```
OPTION BASE 1
DIM Var(5)
FOR I% = 1 TO 5
    INPUT Var(I%)
NEXT I%
END
```
4. Temp(0); Temp(23)
5. 下标
6. Temp(1); Temp(24)
7.
```
Total = 0
FOR I% = 1 TO 24
    Total = Total + Temp(I%)
NEXT I%
Average = Total / 24
PRINT "The average temperature was "; Average
```

8.
```
LowTemp = 999
FOR I% = 1 TO 24
     IF  Temp(I%) < LowTemp THEN
         LowTemp = Temp(I%)
     END IF

NEXT I%
PRINT "The low for the day was "; LowTemp
```

9.
```
LowTime = 0
LowTemp = 999
FOR I% = 1 TO 24
     IF  Temp(I%) < LowTemp THEN
         LowTemp = Temp(I%)
         LowTime = I%
     END IF
NEXT I%
PRINT "At "; LowTime; ":00 the temperature was "; LowTemp
```

10.
```
Count = 0
FOR I% = 1 TO 24
     IF  Temp(I%) < 20 THEN
         Count = Count + 1
     END IF
NEXT I%
PRINT "The temperature fell below 20 "; Count; " hours"
```

自我测验(二)

1.
```
StudentCount = 0
FOR I% = 1 TO 20
     IF  Exam(I%) >= 90 THEN
              StudentCount = StudentCount + 1
     END IF
NEXT I%
PRINT "Total No. of Students with Grades >= 90 is "; StudentCount
```

2.
```
Total = 0
FOR I% = 1 TO 20
     Total = Total + Exam(I%)
NEXT I%
Average = Total / 20
Curve = 80 - Average
IF  Curve < 0 THEN
     Curve = 0
END IF
PRINT "The Curve is "; Curve
```

3.
```
FOR Temp% = 1 TO 24
     PRINT "ENTER TEMPERATURE FOR "; Temp%; ":00 HOURS "
     INPUT Temperature (Temp%)
NEXT Temp%
```

4.
```
HighTemp = - 999
TempX = 1
DO  WHILE TempX <= 24
    IF  Temperature (TempX) > HighTemp THEN
        HighTemp = Temperature (TempX)
        HighTime = TempX
    END IF
    TempX = TempX + 1
LOOP
```
5.
```
TotalTemp = 0
AverageTemp = 0
FOR TempX = 13 TO 24
    TotalTemp = TotalTemp + Temperature (TempX)
NEXT TempX
AverageTemp = TotalTemp / 12
PRINT "Average Temperature for 1 PM-midnight is "; AverageTemp
```
6.
```
OPTION BASE 1
DIM StatePop (50)
FOR IX = 1 TO 50
    PRINT "Enter population for state number "; IX
    INPUT StatePop (IX)
NEXT IX
USAPopulation = 0
FOR IX = 1 TO 50
    USAPopulation = USAPopulation + StatePop (IX)
NEXT IX
PRINT "Total population = "; USAPopulation
END
```
7.
```
CLS
OPTION BASE 1
DIM Dow (366)       '366 was used because 1992 is a Leap Year
FOR DayX = 1 TO 366
    INPUT Dow (DayX)

NEXT DayX
.
.
.
FOR DayX = 1 TO 366
    IF  Dow (DayX) < 1800 THEN
        DowAverage = DowAverage + 1
    END IF
NEXT DayX
PRINT "The Dow fell below 1800 "; DowAverage; " times."
END
```

注释：如果有附加处理，使用数组速是合理的，即是说，只算出Dow低于1800的天数，可不用数组。

自我测验(三)

1.
```
CLS
OPEN "SaleTran.Dat" FOR INPUT AS #1
DIM SalesTotal (1:25)
FOR I% = 1 TO 25
    SalesTotal (I%) = 0
NEXT I%
DO  WHILE NOT EOF(1)
    INPUT #1, SalesPersonNo, TransAmt
    IF  SalesPersonNo > 0 AND SalesPersonNo < 26 THEN
        SalesTotal (SalesPersonNo) = SalesTotal (SalesPersonNo)
         + TransAmt
    ELSE
        PRINT "SalesPersonNo file is in error: "; SalesPersonNo
    END IF
LOOP
FOR I% = 1 TO 25
    PRINT "Total for Salesperson "; I%; "Is "; SalesTotal (I%)
NEXT I%
END
```

2.
```
CLS
SalesTotal = 0
MoreInput$ = "Y"
OPEN "SaleTran.Dat" FOR INPUT AS #1
GOSUB ReadRoutine
HoldSalesNo = SalesPersonNo

GOSUB HeadingRoutine
DO  WHILE MoreInput$ = "Y"
    IF  HoldSalesNo <> SalesPersonNo THEN
        GOSUB  ControlBreak
    END IF
        SalesTotal = SalesTotal + TransAmt
    GOSUB ReadRoutine
LOOP
GOSUB ControlBreak
CLOSE #1
END
ControlBreak:
    LPRINT
    LPRINT TAB(10); "Total Sales";
    LPRINT TAB(25); SalesTotal
    LPRINT
    HoldSalesNo = SalesPersonNo
    SalesTotal = 0
    GOSUB HeadingRoutine
RETURN
ReadRoutine:
    IF  EOF(1) THEN
        MoreInput$ = "N"
    ELSE
        INPUT #1, SalesPersonNo, TransAmt
    END IF
RETURN
```

```
HeadingRoutine:
    LPRINT
    LPRINT TAB(10); "Sales for Salesperson ";
    LPRINT TAB(32); HoldSalesNo
    LPRINT
RETURN
```

3.
```
Ctr = 0
FOR I% = 1 TO 25
    IF  SalesTotal (I%) > 5000 THEN
        Ctr = Ctr + 1
    END IF
NEXT I%
PRINT "The Total No. of Entries > 5000 is: "; Ctr
```

4.
```
Greatest = SalesTotal(1)
FOR I% = 2 TO 25
    IF  SalesTotal(I%) > Greatest THEN
        Greatest = SalesTotal(I%)
    END IF
NEXT I%
PRINT "Largest Sales Amt is: "; Greatest
```

5.
```
OPTION BASE 1
CLS
LPRINT "MONTH"; TAB(15); "TOTAL SALES"
LPRINT "     "; TAB(15); "           "
DIM Sales (12)
ERASE Sales
GOSUB PromptAndGetInput
DO  WHILE SalesAmt > 0
    IF  Month > 0 AND Month < 13 THEN
        Sales (Month) = Sales (Month) + SalesAmt
    ELSE
        PRINT "Month Error: "; Month
    END IF

    GOSUB PromptAndGetInput
LOOP
FOR I% = 1 TO 12
    LPRINT I%;
    LPRINT TAB(15);
    LPRINT USING "$$##,###.##"; Sales (I%)
NEXT I%
LPRINT "End of Report"
END
PromptAndGetInput:
    PRINT "ENTER SALES AMOUNT:"
    INPUT SalesAmt
    PRINT "ENTER MONTH OF SALES (1-12):"
    INPUT Month
RETURN
```

6.
```
CLS
DIM DailyDouble (1:365)
Num = 1
DO  WHILE Num <= 365                'No leap year assumed
    PRINT "Please enter combination for day "; Num
    INPUT Combination
    IF  Combination < 1 OR Combination > 88 THEN
        PRINT "Error in combination.....try again"
    ELSE
        DailyDouble (Num) = Combination
```

```
            Num = Num + 1
        END IF
    LOOP
    END
```

7.
```
FOR I% = 1 TO 365
    IF  DailyDouble(Num) = 63 THEN
        TimesFor63 = TimesFor63 + 1
    END IF
NEXT I%
PRINT "Combination 6-3 won "; TimesFor63; " times last year"
```

8.
```
OPTION BASE 1
DIM Number (25)
FOR I% = 1 to 25
    INPUT Number (I%)
NEXT%
FOR I% = 1 TO 25
    LPRINT I%, Number (I%)
NEXT I%
END
```

9.
```
OPTION BASE 1
DIM Number (25)
FOR I% = 1 TO 25
    INPUT Number(I%)
NEXT I%
FOR I% = 25 TO 1 STEP -1
    LPRINT I%, Number (I%)
NEXT I%
END
```

10.
```
CLS
DIM Number (25)
TotalNumber = 0
AverageNumber =
FOR I% = 1 TO 25
    TotalNumber = TotalNumber + Number (I%)
NEXT I%

AverageNumber = TotalNumber / 25
PRINT "The total is "; TotalNumber
PRINT "The average is "; AverageNumber
```

自我测验(四)

1. 动态
2. $DYNAMIC
3.
```
DIM STATIC Computer (1:100)
DIM DYNAMIC NoRooms (1:150)
DIM DYNAMIC Classes
```
4. F —— ERASE语句将彻底删除任何 DYNAMIC 数组。对于 STATIC 数组，ERASE 语句将数字变量置成 0、字符串变量置成空串。
5. T
6.
```
DIM SocSecNo (1:20)
DIM EmpName$ (1:20)
ERASE SocSecNo, EmpName$
FOR I% = 1 TO 20
    PRINT "Enter Social Security Number: ";
```

```
        INPUT SocSecNo (I%)
        PRINT "Enter Employee Name: ";
        INPUT EmpName$ (I%)
    NEXT I%
```

7.
```
LET I% = 1
PRINT "Enter Social Security Number ";
INPUT SocSecNoIn
DO  WHILE I% <= 20 AND SocSecNo (I%) <> SocSecNoIn
    I% = I% + 1
LOOP
IF  I% > 20 THEN
    PRINT "No Match Found"
ELSE
    PRINT "The Employee Name is "; EmpName$(I%)
END IF
```

8. F —— 所有表格数据都应在进行任何处理之前装入。

9. T

10.
```
PRINT "Enter Model of Car-Type XXX When Done "
INPUT ModelCarIn$
DO  WHILE ModelCarIn$ <> "XXX"
    I% = 1
    DO  WHILE I% <= 10 AND ModelCarIn$ <> ModelCar$(I%)
        I% = I% + 1
    LOOP
    IF  I% > 10 THEN
        PRINT "No Match Found"
    ELSE
        PRINT "Price is "; Price(I%)
    END IF
    INPUT ModelCarIn$
LOOP
PRINT "End"
END
```

自我测验(五)

1.
```
DIM WeightMax (1:20)
DIM DeliveryCharge (1:20)
```

2.
```
FOR I% = 1 TO 20
    PRINT "Please enter maximum weight, delivery charge"
    INPUT WeightMax (I%), DeliveryCharge (I%)
NEXT I%
```

3.
```
GOSUB GetCustomerData
DO  WHILE CustNo <> 0
    Counter% = 1
    DO  UNTIL WeightMax (Counter%) > WeightMarked
        Counter% = Counter% + 1
    LOOP
    Total = Amt + DeliveryCharge (Counter%)
    PRINT CustNo, Amt, DeliveryCharge (Counter%), Total
    GOSUB GetCustomerData
LOOP.
```

注：这里假定EwightMax (20) 取最可能的值 (比如999999)，因此EiwghtMax (20) 一定大于 WeightMarked，故不需进行有效化检查。

4. 并行的。

5. T

6.
```
FOR I% = 1 TO 50
    PRINT "Enter population for state number "; I%
    INPUT StatePop (I%)
NEXT I%
```

7.
```
TotalPop = 0
FOR I% = 1 TO 50
    TotalPop = TotalPop + StatePop (I%)
NEXT I%
PRINT "The Total USA Population is "; TotalPop
```

8.
```
Count = 0
FOR I% = 1 TO 50
    IF  StatePop (I%) > 10000000 THEN
        Count = Count + 1
    END IF
NEXT I%
PRINT "The Total Number of States with Populations_
                > 10,000,000 is "; Count
```

9.
```
HighestPop    0
HighestState = 0
FOR I% = 1 TO 50
    IF  StatePop (I%) > HighestPop THEN
        HighestPop = StatePop (I%)
        HighestState = I%
    END IF
NEXT I%
PRINT "The Highest Population for any state is "; HighestPop
PRINT "The State with the Highest Population is "; HighestState
```

10.
```
LookPop = 1234000
FOR I% = 1 TO 50
    IF  StatePop (I%) = LookPop THEN
        LookState = I%
    END IF
NEXT I%
PRINT "The State with a Population = 1,234,000 is "; LookState
```

自我测验(六)

1.
```
FOR StateNum% = 1 TO 50
    Total = 0
    FOR I% = 1 TO 10
        Total = Total + PopulationFigures (StateNum%, I%)
    NEXT I%
    PRINT "Population for State "; StateNum%; " = "; Total
NEXT StateNum%
```

2.
```
HighPopulation = 0
HighState% = 0
FOR StateNum% = 1 TO 50
    Total = 0
    FOR I% = 1 TO 10
        Total = Total + PopulationFigures(StateNum%, I%)
    NEXT I%
    IF  Total > HighPopulation THEN
        HighState% = StateNum%
        HighPopulation = Total
    END IF
NEXT StateNum%
PRINT "The State with the Largest Population is "; HighState%
```

3.
```
HighPopulation = 0
HighState% = 0
HighCounty% = 0
FOR StateNum% = 1 TO 50
    Total = 0
    FOR I% = 1 TO 10
        IF  PopulationFigures(StateNum%, I%) > _
                HighPopulation THEN
            HighState% = StateNum%
            HighCounty% = I%
            HighPopulation = PopulationFigures(StateNum%, I%)
        END IF
    NEXT I%
NEXT StateNum%
PRINT "The state with the highest county population is "; _
          HighState%
PRINT "The county with the highest population is "; _
          HighCounty%
```

4.
```
OverCount% = 0
FOR StateNum% = 1 TO 50
    FOR I% = 1 TO 10
        IF  PopulationFigures(StateNum%, I%) > 1000000 THEN
            OverCount% = OverCount% + 1
        END IF
    NEXT I%
NEXT StateNum%
PRINT "Number of counties > 1,000,000 people is "; OverCount%
```

5.
```
OverCount% = 0
FOR StateNum% = 1 TO 50
    Total = 0
    FOR I% = 1 TO 10
        Total = Total + PopulationFigures(StateNum%, I%)
    NEXT I%
    IF  Total > 10000000 THEN
        OverCount% = OverCount% + 1
    END IF
NEXT StateNum%
PRINT "The Number of States Over 10,000,000 is "; OverCount%
```

自我测验(七)

1.
```
OPTION BASE 1
DIM StudentDist (30, 3)
```

2.
```
ERASE StudentDist
```

3.
```
OPTION BASE 1
DIM StudentDist (30, 3)
ERASE StudentDist
'************************************************
' This array will have 30 rows each corresponding
' to a course. Each row will consist of three
' minor or column elements-a course id and
' the corresponding total number of males and
' total number of females for that id
'************************************************
PRINT "Enter Course IDs"
FOR I% = 1 to 30

    INPUT StudentDist (I%, 1)
NEXT I%
PRINT "Enter Student ID and sex (1-male, 2-female)"
PRINT "Type 99999, 9 when done"
INPUT StudId, Sex
DO  WHILE StudId <> 99999
    I% = 1
    DO  WHILE I% <= 30 AND StudId <> StudentDist (I%, 1)
        I% = I% + 1
    LOOP
    IF  I% > 30 THEN
        PRINT "No Match on id: "; StudId
    ELSE
        GOSUB Calculate
    END IF
    INPUT StudId, Sex
LOOP
FOR I% = 1 TO 30
    PRINT "Course: "; StudentDist (I%, 1)
    PRINT "Males: "; StudentDist (I%, 2),
    PRINT "Females: "; StudentDist (I%, 3)
NEXT I%
END
Calculate:
    IF  Sex = 1 THEN
        StudentDist (I%, 2) = StudentDist (I%, 2) + 1
    ELSEIF Sex = 2 THEN
        StudentDist (I%, 3) = StudentDist (I%, 3) + 1
```

```
        ELSE
            PRINT "Sex Code Invalid: "; Sex
        END IF
    RETURN
```

4.
```
    FOR I% = 1 TO 30
        Total = StudentDist (I%, 2) + StudentDist (I%, 3)
        PRINT "Course Number "; StudentDist (I%, 1); _
              "has "; Total; " students"
    NEXT I%
```

5.
```
    Total = 0
    FOR I% = 1 TO 30
        Total = StudentDist (I%, 2) + StudentDist (I%, 3)
        IF  Total > 35 THEN
            PRINT "CLASS "; StudentDist (I%, 1); _
                  " Has More Than 35 Students"
        END IF
    NEXT I%
```

本章自测

1. T —— 若读入的是变量，则应确认.

2.
```
    15
    DIM InstructorName$ (1:15)
```

3.
```
    OPTION BASE 1
    DIM FinalExam (5, 8)
```

4.
```
    40
```

5.
```
    FOR Class% = 1 TO 5
        FOR Student% = 1 TO 8
            PRINT "Enter final exam score for Class "; Class%; _
                         "and Student "; Student%
            INPUT FinalExam (Class%, Student%)
        NEXT Student%
    NEXT Class%
```

6.
```
    FOR Class% = 1 To 5
        TotalScore = 0
        FOR Student% = 1 TO 8
            TotalScore = TotalScore + FinalExam (Class%, Student%)
        NEXT Student%
        Average = TotalScore / 8
        PRINT "The average for class "; Class%;
        PRINT "is "; Average
    NEXT Class%
```

7.

```
FOR Class% = 1 TO 5
    Count = 0
    FOR Student% = 1 TO 8
        IF  FinalExam (Class%, Student%) > 90 THEN
            Count = Count + 1
        END IF
    NEXT Student%
    PRINT "Class Number "; Class%; " has ";
    PRINT Count; " students that scored above 90"
NEXT Class%
```

8.

```
FOR Class% = 1 TO 5
    FOR Student% = 1 TO 8
        Total = Total + FinalExam (Class%, Student%)
    NEXT Student%
NEXT Class%
Average = Total / 40
PRINT "The college average is "; Average
```

9.

```
FOR Class% = 1 TO 5
    HighScore = 0
    FOR Student% = 1 TO 8
        IF  FinalExam (Class%, Student%) > HighScore THEN
            HighScore = FinalExam (Class%, Student%)
        END IF
    NEXT Student%
    PRINT "High score for class "; Class%;
    PRINT " is "; Highscore
NEXT Class%
```

10.

```
Count = 0
FOR Class% = 1 TO 5
    FOR Student% = 1 TO 8
        IF  FinalExam (Class%, Student%) = 100 THEN
            Count = Count + 1
            EXIT FOR
        END IF
    NEXT Student%
NEXT Class%
PRINT "No of Classes in which a grade of 100 was reached is ";
      Count
```

第 十 一 章

自我测验(一)

1. T
2. T
3. GOSUB

4. Label; RETURN
5. end ; END
6. CALL
7. SUB 过程名; END SUB
8. T
9. T
10. T

自我测验(二)

1 调用模块中CALL语句后面的语句。
2 T
3 SUB ; END SUB
4 T
5 F——它可以放在任何地方．然而我们建议先编主要的子程序．后编次要的子程序．因而，调用模块应先于被调用模块进行编程。
6 调用子程序或模块，如果在程序的主体，则称之为主模块．
7
   a. 106
   b. 106
   c. 25
   d. 17
8
```
INPUT X
CALL TripleIt (X*3)
:
END
SUB TripleIt (Y)
    Print Y
END SUB
```
9
```
INPUT X
CALL TripleIt (X)
'X now contains a value of X*3
:
END
SUB TripleIt (X)
    X = X*3
    PRINT X
END SUB
```

10.
```
X = 1
Y = 1
PRINT X
PRINT Y
DO  WHILE X < 50
    CALL SumIt (X, Y)
LOOP
END
SUB SumIt (X, Y)
    Z = X + Y
    PRINT Z
    X = Y
    Y = Z
END SUB
```

11.
```
CALL Line1
CALL Line1
CALL Line3
CALL Line4
CALL Line5
CALL Line4
END
'
SUB LINE1
    PRINT "School is almost over and I'm glad"
END SUB
'
SUB Line3
    PRINT "But when I think about final exams"
END SUB
'
SUB Line4
    PRINT "I just get mad"
END SUB
'
SUB Line5
    PRINT "Yes, sir"
END SUB
```

12.
```
GOSUB Line1
GOSUB Line1
GOSUB Line3
GOSUB Line4
GOSUB Line5
GOSUB Line4
END
'
Line1:
    PRINT "School is almost over and I'm glad"
RETURN
'
Line3:
    PRINT "But when I think about final exams"
RETURN
```

```
Line4:
    PRINT "I just get mad"
RETURN
'
Line5:
    PRINT "Yes, sir"
RETURN
```

13 打印如下结果:

```
1  2
2  0
2  0
```

14 打印如下结果:

```
5    15
10   15
```

15 完整的程序如下:

EDIT 窗口

```
CLS
Sum = 0
Answer = 0
PRINT "Module","First","Second","Sum","Answer"
First = 1
Second = 2
PRINT "Main",First,Second,Sum,Answer
CALL SquareSum(First,Second,Sum)
PRINT "Main",First,Second,Sum,Answer
PRINT
CALL SquareSum(Second,First,Answer)
PRINT "Main",First,Second,Sum,Answer
END

SUB SquareSum(First,Second,Sum)
        First = First^2
        Second= Second^2
        Sum = First + Second
        Print "SquareSum",First,Second,Sum,Answer
End SUB
```

RUN 窗口

```
Module      First     Second    Sum     Answer
Main        1         2         0       0
SquareSum   1         4         5       0
Main        1         4         5       0
```

```
SquareSum     16     1     17     0
Main          1      16    5      17
```

16

```
SUB GetNumbers (Num1, Num2)
    PRINT "What is the first number?"
    INPUT Num1
    PRINT "What is the second number?"
    INPUT Num2
END SUB
SUB Switchem (Num1, Num2)
    Hold = Num1
    Num1 = Num2
    Num2 = Hold
END SUB
SUB PrintTheResults (Num1, Num2)
    PRINT "The numbers in reversed order are "; Num1; _
          "and"; Num2
          END SUB
```

17 打印如下结果:

```
12   3   4
55   7   8
 5   7   3   8
```

过程被调用时，变时Y、x及w的值被传给了z、x和w. 由于过程体内对上述几个变量都作了修改，因而具体向过程传送什么值是无关紧要的. 过程执行后，z 中的值 7 传给了 y，x 的值 5 传给了 x，w 中的值 8 保持不变. 在过程调用中的 z 值不受影响，仍维持原来的 3，因为它不在 CALL 语句的参数表中.

18

```
CLS
OPEN "Empl.Dat" FOR OUTPUT AS #1
CALL EnterData(EmployeeName$, PayRate)
DO  WHILE PayRate <> 0
    CALL PayIncrease (PayRate, 0.20)
    PRINT "Employee Name: "; EmployeeName$
    PRINT "New Pay Rate: "; PayRate
    WRITE #1, EmployeeName$, PayRate
    CALL EnterData (EmployeeName$, PayRate)
LOOP
CLOSE #1
PRINT "End of Job"
END
SUB EnterData(EmployeeName$, PayRate)
    PRINT "Enter Employee Name ";
    INPUT EmployeeName$
    PRINT "Enter current pay rate (0 when done) ";
    INPUT PayRate
END SUB
```

```
SUB PayIncrease (PayRate, Increase)
    PayRate = PayRate + Increase * PayRate
END SUB
```

19

```
CLS
OPEN "Empl.Dat" FOR OUTPUT AS # 1
CALL EnterData(EmployeeName$, PayRate, HoursWorked)
DO  WHILE PayRate <> 0
    CALL PayIncrease ((PayRate), 0.15, HoursWorked)
    WRITE #1, EmployeeName$, PayRate * HoursWorked
    CALL EnterData (EmployeeName$, PayRate, HoursWorked)
LOOP
PRINT "End of Job"
CLOSE #1
END
SUB EnterData(EmployeeName$, PayRate, HoursWorked)
    PRINT "Enter Employee Name ";
    INPUT EmployeeName$
    PRINT "Enter pay rate (0 when done) ";
    INPUT PayRate
    PRINT "Enter number of hours worked ";
    INPUT HoursWorked
END SUB
SUB PayIncrease (PayRate, Increase, HoursWorked)
    PayRate = PayRate + Increase * PayRate
    Pay = PayRate * HoursWorked
    PRINT "Employee Name: ", EmployeeName$
    PRINT "Gross Pay with Bonus: ", Pay
END SUB
```

20.

```
CLS
OPEN "Empl.Dat" FOR OUTPUT AS # 1
CALL EnterData(EmployeeName$, PayRate, HoursWorked)
DO  WHILE PayRate <> 0
    CALL PayIncrease (PayRate, 0.15, HoursWorked, Pay)
    PRINT "Employee Name: "; EmployeeName$
    PRINT "Gross Pay: "; Pay
    WRITE #1, EmployeeName$, Pay
    CALL EnterData (EmployeeName$, PayRate, HoursWorked)
LOOP
PRINT "End of Job"
CLOSE #1
END
SUB EnterData(EmployeeName$, PayRate, HoursWorked)
    PRINT "Enter Employee Name ";
    INPUT EmployeeName$
    PRINT "Enter pay rate (0 when done) ";
    INPUT PayRate
    PRINT "Enter number of hours worked ";
    INPUT HoursWorked
END SUB
SUB PayIncrease (PayRate, Increase, HoursWorked, Pay)
    PayRate = PayRate + Increase * PayRate
    Pay = PayRate * HoursWorked
END SUB
```

本章自测

1. 15　25　40
   25　40　15
   25　40　15
2. A B C 的值不同，因为在过程Switch1中它们的值被改变了，并且新的值被传回到了主模块。
3. 值相同是因为在称作Switch2的过程中对A，B，C 所做的修改没有传回主模块。
4. 变参传送。
5. 按值传送。
6. 参数
7. 内相关
8. 层次图
9.

```
CALL DiscountRtn(SaleAmt, 0.10, 0.15, Discount)
•
•
•
SUB DiscountRtn(Sale, Percent1, Percent2, Discount)
    IF  Sale <= 500 THEN
        Discount = Sale * Percent1
    ELSE
        Discount = Sale * Percent2
    END IF
END SUB
```

10.

```
CALL DiscountRtn(SaleAmt, Discount)
•
•
•
SUB DiscountRtn(Sale, Discount)
    IF  Sale <= 500 THEN
        Discount = Sale * 0.10
    ELSE
        Discount = Sale * 0.15
    END IF
END SUB
```

第 十 二 章

自我测验(一)

1. 子程序; 过程; 函数——每一种都可看作是一个模块.
2. 函数返回一个值，然而过程可执行任何操作的集合
3. FN
4. T
5. T
6. T
7. T
8. T
9. T
10. T
11. T
12.

```
DEF FNSumIt(Amt1, Amt2) = Amt1 ^ 2 + Amt2 ^ 2
```

or

```
DEF FNSumIt(Amt1, Amt2)
    FNSumIt = Amt1 ^ 2 + Amt2 ^ 2
END DEF
```

13. F——函数不能改变传递给它的参数值。
14.

```
    Average = FNFindAverage(A, B, C)
    .
    .
    .
END
DEF FNFindAverage = (A + B + C) / 3
```

15.

```
    INPUT N
    Result1 = FNEvenSum(N)
    PRINT "Sum of Even Numbers From 2 to "; N; "Is:"
    PRINT Result1
    .
    .
    .
END
```

```
DEF FNEvenSum(N)
    LET Total = 0
    FOR Integer = 2 TO N STEP 2
        LET Total = Total + Integer
    NEXT Integer
    FNEvenSum = Total
END DEF
```

自我测验(二)

1.

A$ = "School is almost over"

B$ = "glad"

C$ = ""

D$ = "I am glad"

E$ = "I am glad it is almost over"

F = 34

2.

EDIT 窗口

```
CLS
INPUT Number$
AreaCode$ = LEFT$ (Number$, 3)
Exchange1$ = MID$ (Number$, 5, 3)
Last4Digits$ = RIGHT$ (Number$, 4)
PRINT "Area Code = "; AreaCode$
PRINT "Exchange = "; Exchange1$
PRINT "Last 4 digits = ";Last4Digits$
END
```

RUN 窗口

```
?516-560-5716
Area Code = 516
Exchange = 560
Last 4 digit = 5716
```

3. 因为Number$中包含连字符 (-).

自我测验(三)

1. −53

2. 7
3. .4000000059604645
4. 0
5. 45.07
6. 346
7. 47
8. 47
9. 32
10. 100000
11. 72
12. 72
13. −1
14. 1
15. 6
16. turbo basic
17. 16
18. micro
19. ssssssssss(STRING $ 返回"surprise"中首字母的十份)。
20. TURBO BASIC

自我测验(四)

1. 每次执行Test时都将打印0，因为LOCAL属性每次都要复位变量。
2. 8——Number1不受CALL的影响。
3. 2（第一次打印0，第二次打印1，第三次打印2）。
4. STATIC(静态)
5. SHARED(共享)
6. F——在子程序中使用的所有变量都是共享变量。
7. T
8. 每次引用过程或函数时，局部变量的值都被重新初始化。但静态变量将把它的值保存到下一次调用。
9.

```
INPUT X
Code = FNOddEven(X)
IF  Code = 0 THEN
    PRINT "The number "; X; " is even"
ELSE
    PRINT "The number "; X; " is odd"
END IF
END
```

```
'subprograms go here
DEF FNOddEven(X)
    SHARED Code
    LOCAL Y
    Y = X / 2
    IF  INT(Y) = Y THEN
        Code = 0
    ELSE
        Code = 1
    END IF
    FNOddEven = Code
END DEF
```

10.
```
INPUT X
CALL OddEven(X)
.
.
.
END
SUB OddEven(X)
    LOCAL Y, Code
    Y = X / 2
    IF  INT(Y) = Y THEN
        Code = 0
    ELSE
        Code = 1
    END IF
    IF  CODE = 0 THEN
        PRINT "The number "; X; " is even"
    ELSE
        PRINT "The number "; X; " is odd"
    END IF
END SUB
```

本章自测

1.
```
s
us
mus
amsus
tamus
otamus
potamus
opotamus
popotamus
ppopotamus
ippopotamus
Hippopotamus
```

2.
```
H
Hi
Hip
Hipp
Hippo
Hippop
Hippopo
Hippopot
Hippopota
Hippopotam
Hippopotamu
Hippopotamus
```

3.
```
'Translate English to Pig Latin
PigLatin$ = ""                  'Initialize pig latin string
INPUT English$                  'Read English text
Length = LEN(English$)            'Get its length
Start = 1                         'Point to start of first word
DO  WHILE Start < Length          'While more text remains
    J = Instr(Start, English$, " ")  'Blank is the end of word
    GOSUB WordTrans               'Translate word
    Start = J + 1                 'Point to start of next word
LOOP
PRINT "Pig Latin translation:"
PRINT PigLatin$                   'Translation done, print it
END
WordTrans:          'Translate a word to pig latin routine
    FirstChar$ = MID$(English$, Start, 1)     'First character
    'Rest of word
    RestofWord$ = MID$(English$, Start + 1, J-Start-1)
    PigLatinWord$ = RestofWord$ + FirstChar$ + "ay"
    PigLatin$ = PigLatin$ + PigLatinWord$
RETURN
```

4.
```
'Guess the Number Game
DEF FNRndint(A, B) = A + INT ((B - A + 1) * RND)
PRINT "Guess a numbe  from 1 to 100."
PRINT "You get 10 guesses - Good Luck!"
Number = FNRndint(1, 100)
FOR Trial = 1 TO 10
    INPUT "Guess "; Guess

     IF  Guess = Number THEN
         PRINT "Congratulations! You guessed the number in "; _
              Trial; "guesses"
         EXIT FOR
     ELSEIF
         Guess > Number THEN
             PRINT "Too high"
     ELSE
         PRINT "Too low"
     END IF
NEXT Trial
PRINT "Sorry, you used 10 guesses. The number was "; Number
END

RUN
Guess a number from 1 to 100.
You get 10 guesses - Good Luck!
Guess? 50
Too high
Guess? 25
Too low
```

B22 8-32 (2)

```
Guess? 37
Too high
Guess? 31
Too high
Guess? 28
Too High
Guess? 26
Congratulations! You guessed the number in 6 guesses
```

5.
```
BOB 226
RICHARD 15
```

6.
```
 22.62
182.83
  7.40
-15.64
```

7.
```
 32.46
 72.86
  6.30
-27.28
675.
```

8. a. 13.8
   b. 6.3
   c. 39
   d. 4
   e. me

9. 3

10. ******

11. 1 2 3 4 5

12. 4

13.
```
C
CD
CDE
```

14.
```
 ******************************************
  This program simulates the game of craps
 ******************************************
CLS
NumWins = 0
NumLosses = 0
INPUT "Number of games of craps to play";N
FOR Games = 1 to N
        Total = INT (6 * RND) + INT (6 * RND) + 2
        Print "Total: ",Total
        IF Total = 7 OR Total = 11 THEN
                NumWins = NumWins + 1
                PRINT "That's a win"
        ELSEIF Total = 2 OR Total = 3 OR Total = 12 THEN
                PRINT "That's a loss"
                NumLosses = NumLosses + 1
        ELSE
                CALL KeepRolling (Total,NumWins,NumLosses)
        END IF
NEXT Games
PRINT "Number of Wins"; NumWins
PRINT "Number of Losses";NumLosses
END
SUB KeepRolling (Total,NumWins,NumLosses)
        PRINT "Roll again"
        LET Point1 = Total
        Total = INT (6 * RND) + INT (6 * RND) + 2
        PRINT "Next dice simulation is ";Total
                DO WHILE Total <> Point1 and Total <> 7
                        Total = INT (6 * RND) + INT (6 * RND) + 2
                        PRINT "Next dice simulation is ";Total
                LOOP
```

22A8-32②

```
            IF Total = Point1 THEN
                    NumWins = NumWins + 1
                    PRINT "That's a win"
            ELSE
                    NumLosses = NumLosses + 1
                    PRINT "That's a loss"
            END IF
END SUB
```

15.
```
 ******************************************
  This program converts upper case letters
  to lower case letters
 ******************************************
CLS
INPUT "Enter Text: ";Text$
Length = LEN (Text$)
RevisedText$ = ""
FOR I% = 1 TO LENGTH
        C$ = MID$ (Text$, I%,1)
        IF C$ >= "A" AND C$ <= "Z" THEN
                C$ = CHR$ (ASC(C$) + 32)
 'CHR$ (ASC(C$) + 32 is the lower case equivalent of C$
        END IF
        RevisedText$ = RevisedText$ + C$
NEXT I%
PRINT "Version of the text with upper case letters:
PRINT Text$
PRINT
PRINT "Lower case version of text: "
PRINT RevisedText$
```

第 十 三 章

自我测验(一)

1. 主文件
2. 记录; 记录; 字段
3. 细目.
4. 旧主文件，它在前一修改周期中为最新的; 细目文件; 新主文件，它合并包含有旧主文件中的数据和细目文件中的数据 .
5. T
6. 旧主文件(在说明中为OldMstr文件)
7. 正常修改; 将细目文件中的数据加到主文件中去，得到一新主文件记录，读入这条来自旧主文件和细目文件的记录.
8. 不修改，直接从旧主文件主记录中建立新主文件记录，读入这条来自旧主文件的记录.
9. 新账或者出错. 如果认为这是一笔新账，直接从细目记录中建立主记录; 如果我们认为这一记录出错了，将打印出错误信息. 在任何情况下，都将读入细目记录.
10. 更换在细目记录中的关键字段(在说明中为TAcctNo)，用尾部数值如9'S来代替，任意顺序下 TAcctNo 和实际的 MAcctNo 相比较都将产生“>”条件. 所以将执行 NoUpdate 过程，旧主文件记录将写到新主文件记录中去，其它的旧主文件记

录将读入.

自我测验(二)

1. 编码字段
2. F–对每条主记录可能有许多细目，但每条记录上只有单个细目记录的修改过程与每条主记录有多个细目记录的修改过程是不相同的.
3. F–若这一细目记录是用于建立新记录时，可能与主文件不相匹配.
4. TYPE
5. 已排序
6.

```
OPEN "Student.Dat" FOR OUTPUT AS #1
CALL EnterData (SocSecNo, StudentName$, Units)
DO  WHILE SocSecNo <> 0
    WRITE #1, SocSecNo, StudentName$, Units
    CALL EnterData(SocSecNo, StudentName$, Units)
LOOP
CLOSE #1
PRINT "End of Run"
END
SUB EnterData(SocSecNo, StudentName$, Units)
    PRINT "Enter Social Security Number, Student Name, Units"
    INPUT SocSecNo, StudentName$, Units
END SUB
```

7.

```
IF  MSocSecNo = TSocSecNo THEN
    MUnits = MUnits + TUnits
    WRITE #3, MSocSecNo, MStudentName$, MUnits
    CALL GetMaster(MSocSecNo, MStudentName$, MUnits)
    CALL GetTrans(TSocSecNo, TStudentName$, TUnits)
ELSEIF TSocSecNo > MSocSecNo THEN
    CALL GetMaster(MSocSecNo, MStudentName$, MUnits)
ELSE
    PRINT "Student not on file "; TSocSecNo
    CALL GetTrans(TSocSecNo, TUnits)
END IF
```

8.

```
IF  MSocSecNo = TSocSecNo THEN
    MUnits = MUnits + TUnits
    WRITE #3, MSocSecNo, MStudentName$, MUnits
    CALL GetMaster(MSocSecNo, MStudentName$, MUnits)
    CALL GetTrans(TSocSecNo, TStudentName$, TUnits)
ELSEIF TSocSecNo > MSocSecNo THEN
    CALL GetMaster(MSocSecNo, MStudentName$, Munits)
ELSE
    WRITE #3, TSocSecNo, TStudentName$, TUnits
    CALL GetTrans(TSocSecNo, TStudentName$, TUnits)
END IF
```

9. T

10. T

自我测验(三)

1. 可变化;固定。
2. 带定界符顺序文件
3. T
4. 因为顺序文件大多数常为纯ASCII文件，所以通常能被移植;而随机文件正相反，用不同的程序和／或软件包来建立不同的随机文件，所以一般是不能移植的。
5. OPEN ; LEN
6. F-顺序文件必须在OPEN语句中指出用作输入(INPUT)或还是用作输出(OUPTUT)。
7. T
8.
```
OPEN "Student.Dat" AS #1 LEN = 28
FILED #, 2 AS StudentId$, 20 AS StudentName$,_
          2 AS Units$, 4 AS GPA$
```
9.
```
StudentId$ = MKI$(StudentName%)
Unit$ = MKI$(UnitsComplete%)
GPA$ = MKS$(GPA%)
```
10.
```
RSET StudentId$ = MKI$(StudentName%)
RSET Unit$ = MKI$(UnitsComplete%)
RSET GPA$ = MKS$(GPA%)
```
注意: LSET 用于字符串变量，而 RSET 用于数值变量。

自我测验(四)

1. F-STR$ 函数不能用于随机文件中将数值转换到字段里。
2. F-随机文件中的所有字段，却必须用RSET或者LSET函数赋值。
3. T
4. PUT
5. T
6. 如果采用一个非连续性的关键字段来确定随机文件中的记录位置,文件正常数据记录中将分配有空记录。
7. 4
8. 2

9.
```
OPEN "Inventory.Dat" AS #1 LEN = 28
FIELD #1, 2 AS InventoryNo$, 20 AS ItemDesc$_
          2 AS NoOnHand$, 4 AS ItemPrice$
PRINT "Entér Number of Records";
INPUT NoRec%
FOR I% = 1 TO NoRec%
    CALL GetData(InventoryNoIn%, ItemDescIn$, NoOnHandIn%, _
                 ItemPriceIn)
    CALL CreateRandomRec(InventoryNoIn%, ItemDescIn$, _
                         NoOnHandIn%, ItemPriceIn)
NEXT I%
CLOSE #1
PRINT "End of Job"
END

SUB GetData(InventoryNoIn%, ItemDescIn$, NoOnHandIn%, _
            ItemPriceIn)
    PRINT "Enter Inventory Number: ";
    INPUT InventoryNoIn%
    PRINT "Enter Item Description: ";
    INPUT ItemDescIn$
    PRINT "Enter Number on Hand: ";
    INPUT NoOnHandIn%
    PRINT "Enter Item Price: ";
    INPUT ItemPriceIn
END SUB
SUB CreateRandomRec(InventoryNoIn%, ItemDescIn$, _
                    NoOnHandIn%, ItemPriceIn)
    LSET InventoryNo$ = MKI$ (InventoryNoIn%)
    RSET ItemDesc$ = ItemDescIn$
    LSET NoOnHand$ = MKI$ (NoOnHandIn%)
    LSET ItemPrice$ = MKS$ (ItemPriceIn)
    PUT #1, I%
END SUB
```

10.
```
OPEN "Inventory.Dat" AS #1 LEN = 28
FIELD #1, 2 AS InventoryNo$, 20 AS ItemDesc$, _
          2 AS NoOnHand$, 4 AS ItemPrice$
PRINT "Enter -9999 for Inventory Number when done"
PRINT "Enter Inventory Number: ";
INPUT InventoryNoIn%
DO  WHILE InventoryNoIn% <> -9999
    CALL GetData(ItemDescIn$, NoOnHandIn%, ItemPriceIn)
    CALLCreateRandomRec(InventoryNoIn%, ItemDescIn$, _
                        NoOnHandIn%, ItemPriceIn)
LOOP
CLOSE #1
PRINT "End of Job"
END
SUB GetData(ItemDescIn$, NoOnHandIn%, ItemPriceIn)
    PRINT "Enter Item Description: ";
    INPUT ItemDescIn$
    PRINT "Enter Number on Hand: ";
    INPUT NoOnHandIn%
    PRINT "Enter Item Price: ";
    INPUT ItemPriceIn
END SUB
```

```
SUB CreateRandomRec(InventoryNoIn%, ItemDescIn$, _
                    NoOnHandIn%, ItemPriceIn)
    LSET InventoryNo$ = MKI$ (InventoryNoIn%)
    RSET ItemDesc$ = ItemDescIn$
    LSET NoOnHand$ = MKI$ (NoOnHandIn%)
    LSET ItemPrice$ = MKS$ (ItemPriceIn)
    PUT #1, InventoryNoIn%
    PRINT "Enter Inventory Number: ";
    INPUT InventoryNoIn%
END SUB
```

自我测验(五)

1. T
2. F—可能从一没有确定是用作INPUT还是用作OUTPUT的随机文件中读和写 /
3. GET
4. T
5. LOF(文件长度)
6. F——删除记录用不活动来标记．即:建立一个活动码字段，其一个值代表活动记录，另一值表示不活动记录；所以，实际上是通过变化活动码字段的值来指明删除记录．因此，此记录仍就在文件上．
7.

```
OPEN "Inventory.Dat" AS #1 LEN = 29
FIELD #1, 2 AS InventoryNo$, 20 AS ItemDesc$, _
          2 AS NoOnHand$, 4 AS ItemPrice$, 1 AS ActivityFlag$
RecNo% = 1
DO  WHILE RecNo% <= LOF(1) / 29
    GET #1, RecNo%
    IF  ActivityFlag$ <> "I" THEN
        PRINT "Inventory Number: "; CVI(InventoryNo$)
        PRINT "Item Description: "; ItemDesc$
        PRINT "Number on Hand: "; CVI(NoOnHand$)
        PRINT "Item Price: "; CVS(ItemPrice$)
    END IF
    RecNo% = RecNo% + 1
LOOP
```

8.

```
CLS
OPEN "Inventory.Dat" AS #1 LEN = 29
FIELD #1, 2 AS InventoryNo$, 20 AS ItemDesc$, _
          2 AS NoOnHand$, 4 AS ItemPrice$, 1 AS ActivityFlag$
PRINT "Enter Inventory Number ";
INPUT InventoryNoIn%
DO  WHILE InventoryNoIn% <> -9999
    GET #1, InventoryNoIn%
    PRINT "Item Description: "; ItemDesc$
    PRINT "Number on Hand: "; CVI(NoOnHand$)
```

```
        PRINT "Item Price: "; CVS(ItemPrice$)
        PRINT "Enter Inventor  Number (-9999 When Done):";
        INPUT InventoryNoIn%
    LOOP
    CLOSE #1
    PRINT "End of Job"
    END
```

9

```
    CLS
    OPEN "Inventory.Dat" AS #1 LEN = 29
    FIELD #1, 2 AS InventoryNo$, 20 AS ItemDesc$, _
              2 AS NoOnHand$, 4 AS ItemPrice$, 1 AS ActivityFlag$
    CALL EnterData (InventoryNoIn%)
    DO  WHILE InventoryNoIn% <> -9999
        GET #1, InventoryNoIn%
        PRINT "Item Description: "; ItemDesc$
        PRINT "Number on Hand: "; CVI(NoOnHand$)
        PRINT "Item Price: "; CVS(ItemPrice$)
        PRINT "Is this item to be deleted? (Y or N) "
        INPUT Reply$
        IF  Reply$ = "Y" THEN
            LSET ActivityFlag$ = "I"
            PUT #1, InventoryNoIn%
        ELSE
            PRINT "Item not deleted"
        END IF
        CALL EnterData (InventoryNoIn%)
    LOOP
    CLOSE #1
    PRINT "End of Job"
    END
    SUB EnterData (InventoryNoIn%)
        PRINT "Enter Inventory Number (-9999 When Done):";
        INPUT InventoryNoIn%
    END SUB
```

10.

```
    CLS
    OPEN "Inventory.Dat" AS #1 LEN = 29
    FIELD #1, 2 AS InventoryNo$, 20 AS ItemDesc$, _
           2 AS NoOnHand$, 4 AS ItemPrice$, 1 AS ActivityFlag$
    CALL EnterData (InventoryNoIn%)
    DO  WHILE InventoryNoIn% <> -9999
        PRINT "Item Number "; InventoryNoIn%; " to be added or";
        PRINT " displayed? (Enter A or D) ";
        INPUT Reply$
        IF  Reply$ = "D" THEN
            GET #1, InventoryNoIn%
            PRINT "Item Description: "; ItemDesc$
            PRINT "Number on Hand: "; CVI(NoOnHand$)
            PRINT "Item Price: "; CVS(ItemPrice$)
        ELSEIF Reply$ = "A" THEN
            PRINT "Enter Item Description ";
            INPUT ItemDescIn$
            PRINT "Enter Number on hand ";
            INPUT NoOnHandIn%
            PRINT "Enter Item Price ";
            INPUT ItemPriceIn
            LSET InventoryNo = MKI$(InventoryNoIn%)
            LSET ItemDesc$ = ItemDescIn$
            LSET NoOnHand$ = MKI$(NoOnHand%)
            LSET ItemPrice$ = MKS$(ItemPriceIn)
            LSET ActivityFlag$ = "A"
```

```
        PUT #1, InventoryNoIn%
        PRINT "Item added to file"
    ELSE
        PRINT "Error in input"
    END IF
    CALL EnterData (InventoryNoIn%)
LOOP
CLOSE #1
PRINT "End of Job"
END
SUB EnterData (InventoryNoIn%)
    PRINT "Enter Inventory Number (-9999 When Done):";
    INPUT InventoryNoIn%
END SUB
```

本章自测

1. 顺序文件中记录必须按顺序存取，而随文件中记录可直接存取，顺序文件中必须指明是用作 INPUT 还是用作 OUTPUT，而随机文件可在同一程序中既可用作输入，又可用作输出.
2. 数据的完整性和有效性.
3. 批处理，交互处理.
4. 旧主文件，细目文件，新主文件.
5. 旧主文件.
6. 散列算法.
7. MKI$ 用来将整型数转换成存贮随机文件中2字节值.
   MKS$ 用来将单精度数转换成存贮随机文件中 4 字节值.
   MKD$ 用来将双精度数转换成存贮随机文件中 8 字节值.
8. 如果在GET语句中省略了记录编号，则将存取文件中的下一条记录.
9. FIELD #1, 2 AS CustNo$, 20 AS CustName$,_
   20 AS Address$, 4 AS Balance$
   整型数，如顾客编号 CustNo 是 2 字节；单精度数，如结余 Balance，是 4 字节。
10. LSET CustNo$ = MKI$(CustNoIn%)
    LSET CustName$ = CustNameIn
    LSET Address$ = AddressIn
    LSET Balance$ = MKS$(BalanceIn)
    PUT #1, CustNoIn%

第 十 四 章

自我测验(一)

1. 80，25
2. 象素
3. SCREEN
4. CLS 或SCREEN
5. 馅饼图
6. F——CIRCLE 语句，绘制其大小相应于所占百分比的扇行。
7. F——LOCATE指令用于光标定位。
8. T
9.

```
    SCREEN 2
    COLOR1 = 1
    Total = 0
    X = 320
    Y = 100
    Pi = 3.14159
    A2 = -.00001
    Radius = 150
    DIM Percents(1:6)
    FOR I% = 1 TO 6
        READ Percents(I%)
        Total = Total + Percents(I%)
    NEXT I%
    FOR I% = 1 TO 6
        A1 = A2
        A2 = A1 - Percents(I%) / Total * 2 * Pi
        CIRCLE (X, Y), Radius, Color1, A1, A2
    NEXT I%
    DATA 20, 15, 18, 15, 22, 10
END
```

10.

```
    SCREEN 2
    COLOR1 = 1
    Total = 0
    X = 320
    Y = 100
    Pi = 3.14159
    A2 = -.00001
    Radius = 150
    DIM Percents(1:6)
    FOR I% = 1 TO 6
        READ Percents(I%)
        Total = Total + Percents(I%)
    NEXT I%
    FOR I% = 1 TO 6
        A1 = A2
        A2 = A1 - Percents(I%) / Total * 2 * Pi
```

```
        CIRCLE (X, Y), Radius, Color1, A1, A2
    NEXT I%

    DATA 20, 15, 18, 15, 22, 10
    LOCATE  9, 51 :PRINT "A"
    LOCATE 15, 55 :PRINT "B"
    LOCATE 17, 42 :PRINT "C"
    LOCATE 15, 32 :PRINT "D"
    LOCATE 11, 32 :PRINT "E"
    LOCATE  9, 38 :PRINT "F"
END
```

自我测验(二)

1. WINDOW
2. VIEW
3. LOCATE
4. 象素为行和列的交叉点，可能填入并组成图形（象素— Pixel 是 Picture Element 的缩写）.
5.
```
SCREEN 2
LINE (100, 105) - (319, 0), , B
END
```
6.
```
SCREEN 2
LINE (100, 165) - (319, 0), , B
END
```
7.
```
SCREEN 2
GOSUB GetData
WINDOW (0, 0) - (5, 300)
VIEW (400, 7) - (620, 80)
FOR I% = 1 TO 4
    LINE (I%-1, 0) - (I%, WEEK(I%)), , BF
NEXT I%
END

GetData:
    OPTION BASE 1
    DIM WEEK(4)
    FOR I% = 1 TO 4
        READ WEEK(I%)
    NEXT I%
    DATA 150, 225, 75, 110
RETURN
```
8.
```
SCREEN 2
GOSUB GetData
WINDOW (0, 0) - (5, 300)
VIEW (400, 7) - (620, 80)
FOR I% = 1 TO 4
    LINE (I% - 1, 0) - (I%, WEEK(I%)), , BF
NEXT I%
END
GetData:
    OPTION BASE 1
    DIM WEEK(4)
```

```
    FOR I% = 1 TO 4
        READ WEEK(I%)
    NEXT I%
    DATA 150, 225, 75, 110
RETURN
```

9.
```
SCREEN 2
GOSUB GetData
WINDOW (0, 0) - (5, 300)
VIEW (400, 7) - (620, 80)
FOR I% = 1 TO 4
    LINE (I% - 1, 0) - (I%, WEEK(I%)), , BF
NEXT I%
GOSUB AddLabels
END
GetData:
    OPTION BASE 1
    DIM WEEK(4)
    FOR I% = 1 TO 4
        READ WEEK(I%)
    NEXT I%
    DATA 150, 225, 75, 110
RETURN
AddLabels:
    LOCATE 5, 7 : PRINT "WEEK 1 SALES = ";
    PRINT USING "***"; WEEK(1)
    LOCATE 6, 7 : PRINT "WEEK 2 SALES = ";
    PRINT USING "***"; WEEK(2)
    LOCATE 7, 7 : PRINT "WEEK 3 SALES = ";
    PRINT USING "***"; WEEK(3)
    LOCATE 8, 7 : PRINT "WEEK 4 SALES = ";
    PRINT USING "***"; WEEK(4)
RETURN
```

10.
```
SCREEN 2
GOSUB GetData
WINDOW (0, 0) - (5, 300)
VIEW (400, 7) - (620, 80)
FOR I% = 1 TO 4
    LINE (I% - 1, 0) - (I%, WEEK(I%)), , BF
NEXT I%
GOSUB AddLabels
END


GetData:
    OPTION BASE 1
    DIM WEEK(4)
    FOR I% = 1 TO 4
        READ WEEK(I%)
    NEXT I%
    DATA 150, 225, 75, 110
RETURN
AddLabels:
    LOCATE 5, 7 : PRINT "WEEK 1 SALES = ";
    PRINT USING "***"; WEEK(1)
    LOCATE 6, 7 : PRINT "WEEK 2 SALES = ";
    PRINT USING "***"; WEEK(2)
    LOCATE 7, 7 : PRINT "WEEK 3 SALES = ";
    PRINT USING "***"; WEEK(3)
    LOCATE 8, 7 : PRINT "WEEK 4 SALES = ";
    PRINT USING "***"; WEEK(4)
    LOCATE 13, 34 : PRINT "WEEKLY SALES"
```

```
        LOCATE 7, 39 : PRINT "SALES"
        LOCATE 12, 60 : PRINT "WEEK"
    RETURN
```

自我测验(三)

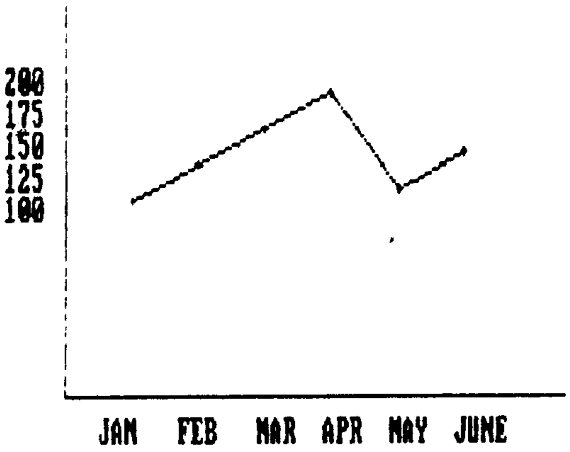
1. F—线条图是一种极好的趋势描述方法.
2. F—馅饼图比线条图较适合描述百分比.
3. LOCATE
4. LINE

注意: 5至10题的解答中，参数可能要有所变化.

5.
```
LINE (15, 30) - (50, 75)
```
6.
```
CIRCLE (15, 30), .2
CIRCLE (50, 75), .2
```
7.
```
SCREEN 2
WINDOW (0, 199) - (319, 0)
LINE (120, 90) - (120, 160)
LINE (20, 160) - (120, 160)
LINE (20, 160) - (120, 90)
END
```
8.
```
SCREEN 1
LINE (100, 75) - (200, 125), , B
LINE (100, 75) - (200, 125)
LINE (200, 75) - (100, 125)
END
```
9.
```
SCREEN 2
WINDOW (0, 199) - (319, 0)
DIM MONTH(1:6)
GOSUB EnterData
CLS
GOSUB DrawChart
END
EnterData:
    FOR I% = 1 TO 6
        PRINT "Enter units sold for month "; I%
        INPUT MONTH (I%)
    NEXT I%
RETURN
DrawChart:
    FOR I% = 1 TO 6
        x1 = 70 + 20 * I%
        y1 = 80 + .2 * MONTH (I%)
        x2 = 70 + 20 * (I% + 1)
        y2 = 80 + .2 * MONTH (I% + 1)
        CIRCLE (x1, y1), .2
        LINE (x1, y1) - (x2, y2)
    NEXT I%
    CIRCLE (x2, y2), .2
RETURN
```

10.
```
SCREEN 2
WINDOW (0, 199) - 319, 0)
DIM MONTH(1:6)
GOSUB EnterData
CLS
LINE (70, 60) - (70, 160)
LINE (70, 60) - (220, 60)
GOSUB DrawChart
GOSUB AddLabels
END
EnterData:
    FOR I% = 1 TO 6
        PRINT "Enter units sold for month "; I%
        INPUT MONTH (I%)
    NEXT I%
RETURN
DrawChart:
    FOR I% = 1 TO 5
        x1 = 70 + 20 * I%
        y1 = 80 + .3 * MONTH (I%)
        x2 = 70 + 20 * (I% + 1)
        y2 = 80 + .3 * MONTH (I% + 1)
        CIRCLE (x1, y1), .2
        LINE (x1, y1) - (x2, y2)
    NEXT I%
    CIRCLE (x2, y2), .2
RETURN
AddLabels:
    LOCATE 12, 13 : PRINT 100
    LOCATE 11, 13 : PRINT 125
    LOCATE 10, 13 : PRINT 150
    LOCATE  9, 13 : PRINT 175
    LOCATE  8, 13 : PRINT 200
    LOCATE 19, 21 : PRINT "JAN"
    LOCATE 19, 27 : PRINT "FEB"
    LOCATE 19, 33 : PRINT "MAR"
    LOCATE 19, 38 : PRINT "APR"
    LOCATE 19, 43 : PRINT "MAY"
    LOCATE 19, 48 : PRINT "JUNE"
RETURN
```

图形结果为:



自我测验(四)

1. PSET
2. PSET STEP
3. T
4. T
5. DRAW
6. F——SCREEN2 指定高分辨率图形模式，而中分辨率模式是SCREEN 1.
7.

```
SCREEN 1, 0
COLOR 4, 1
FOR I% = 20 TO 80
    FOR J% = 20 TO 170
        PSET (I%, J%), 1
    NEXT J%
NEXT I%
END
```

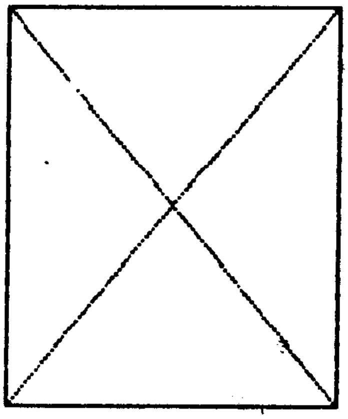
8.

```
DRAW "R60D50L60U50"
```

9.

```
SCREEN 1, 1
PSET (50, 125)
DRAW "E50; F50; L100"
PSET (50, 125)
DRAW "E50; H50; D100"
PSET (150, 125)
DRAW "H50; E50; D100"
PSET (150, 25)
DRAW "G50; H50; R100"
END
```

其结果如下



10. （注意：颜色选择要有所变动）

```
SCREEN 1, 0
FOR I% = 30 TO 70
    PSET (40, I%), 1
NEXT I%
FOR J% = 40 TO 80
    PSET (J%, 70), 2

NEXT J%
FOR I% = 30 TO 70
    PSET (80, I%), 1
NEXT I%
FOR J% = 40 TO 80
    PSET (J%, 30), 2
NEXT J%
END
```

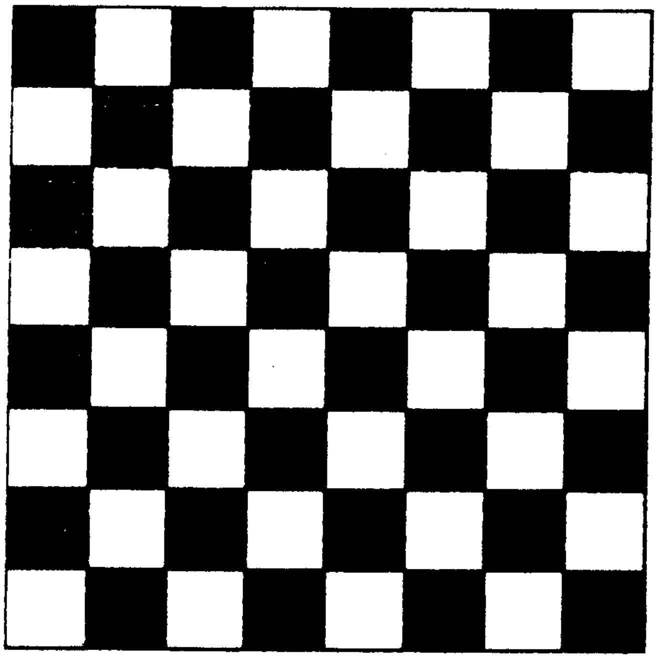
本章自测

1. F——在高分辨率图形模式下，可编址的象素比中分辨率图形模式下的多。
2. T
3. T
4. F——LOCATE 语句指定字符位置；而PSET 语句用来指定精确的象素位置。
5. 32
6. SCREEN 2
7. BEEP
8. PSET
9. T
10.

```
  ************************************************
   This program draws a checkerboard on the screen
   in medium resolution graphics.  The user is
   prompted for the height of the squares
  ************************************************
SCREEN 1
COLOR 0,0
CLS
N = 20   20 is the height in pixels of each square
XSTEP = N * 1.2
YSTEP = N
Y1 = 10   Y1 is the Y coordinate of starting point of checkerboard
RED% = -1 ' RED% is 0 or 1 simulating red or black
FOR ROW% = 1 TO 8
        X1 = 80   X1 is the X coordinate of starting point of board
        FOR COL% = 1 TO 8
                IF RED% <> 0 THEN
                        LINE (X1,Y1) - STEP (XSTEP,YSTEP),2,BF
                        RED% = 0
                ELSE
                        LINE (X1,Y1) - STEP (XSTEP,YSTEP),2,B
                        RED% = -1
                END IF
                        X1 = X1 + XSTEP
        NEXT COL%
Y1 = Y1 + YSTEP
IF RED% <> 0 THEN
        RED% = 0
```

```
ELSE
        RED% = -1
END IF
NEXT ROW%
END
```

其结果为:



附　　录

本部分内容

附录 A　Turbo Basic 保留字
附录 B　Turbo Basic 字符集
附录 C　ASCII(美国信息交换标准代码)表
附录 D　Turbo Basic 编辑命令表
附录 E　Turbo Basic 错误信息
附录 F　Turbo Basic 指令集

附录 A　Turbo Basic 保留字

$COM1	CDBL	DEFSTR	GET$
$COM2	CEIL	DELAY	GOSUB
$DEBUG	CHAIN	DIM	GOTO
$DYNAMIC	CHDIR	DO	HEX$
$ELSE	CHR$	DRAW	IF
$ENDIF	CINT	DYNAMIC	IMP
$EVENT	CIRCLE	ELSE	INCR
$IF	CLEAR	ELSEIF	INKEY$
$INCLUDE	CLNG	END	INLINE
$INLINE	CLOSE	ENDIF	INP
$LIST	CLS	ENVIRC	INPUT
$OPTION	COLOR	ENVIRON$	INPUT#
$SEGMENT	COM	EOF	INPUT#
$SOUND	COMMAND	EQV	INPUT$
$STACK	COMMON	ERADP	INSTAT
$STATIC	COS	ERASE	INSTR
ABS	CSNG	ERDEV	INT
ABSOLUTE	CSRLIN	ERDEV$	INTERRUPT
AND	CVD	ERL	IOCTL
APPEND	CVI	ERR	IOCTL$
AS	CVL	ERROR	KEY
ASC	CVMD	EXIT	KILL
AT	CVMS	EXP	LBOUND
ATN	CVS	EXP10	LCASE$
BASE	DATA	EXP2	LEFT$
BEEP	DATE$	FIELD	LEN
BIN$	DECR	FILES	LET
BINARY	DEF	FIX	LINE
BLOAD	DEFDBL	FN	LIST
BSAVE	DEFINT	FOR	LOC
CALL	DEFLNG	FRE	LOCAL
CASE	DEFSNG	GET	LOCATE

LOF	OUTPUT	SAVE	TROFF
LOG	PAINT	SCREEN	TRON
LOG10	PALETTE	SEEK	UBOUND
LOG2	PEEK	SEG	UCASE$
LOOP	PEN	SELECT	UNTIL
LPOS	PLAY	SERVICE	USING
LPRINT	PMAP	SGN	USR
LPRINT#	POINT	SHARED	USR0
LSET	POKE	SHELL	USR1
MEMSET	POS	SIN	USR2
MID$	PRESET	SOUND	USR3
MKD$	PRINT	SPACE$	USR4
MKDIR	PRINT#	SPC	USR5
MKI$	PSET	SQR	USR6
MKL$	PUT	STATIC	USR7
MKMD$	PUT$	STEP	USR8
MKMS$	RANDOM	STICK	USR9
MKS$	RANDOMIZE	STOP	VAL
MOD	READ	STR$	VARPTR
MTIMER	REG	STRIG	VARPTR$
NAME	REM	STRING#	VARSEG
NEXT	RESET	SUB	VIEW
NOT	RESTORE	SWAP	WAIT
OCT$	RESUME	SYSTEM	WEND
OFF	RETURN	TAB	WHILE
ON	RIGHT$	TAN	WIDTH
OPEN	RMDIR	THEN	WINDOW
OPTION	RND	TIME$	WRITE
OR	RSET	TIMER	WRITE#
OUT	RUN	TO	XOR

附录 B　Turbo Basic 字符集

在 Turbo Basic 中，可以用 A～Z，a～z，0～9 等字母数字组成标识符(标号、变量、过程和函数的名字)；0～9，+，.，E，e，D，d 可用于组成数值常数。下面的符号在 Turbo Basic 中有特定的含义:

符号	名称	功　能
=	等号	赋值操作，关系运算符
+	加号	相加，字符串的连结符
−	减号	相减，角号
*	星号	乘法运算符
/	斜杠	除法运算符
＼	反斜杠	整除运算符
^	托字符	乘方运算符
%	百分号	整型说明符
&	和号	长整型说明符
!	叹号	单精度说明符
#	井号	双精度说明符
$	美元符	字符串说明符；元语句的前缀
( )	圆括号	用于函数和过程的参数传递、数组、要优先处理的表达式
[ ]	方括号	只用于数组
	空格	分隔符
,	逗号	通用的定界符
.	句号	小数点，文件名与扩展名之间的分隔符
'	单引号	注释的定界符
;	分号	通用的定界符
:	冒号	语句定界符
?	问号	PRINT 的代替符
<	小于	关系运算符
>	大于	关系运算符
"	双引号	字符中的定界符
_	下划线	续行符

附录 C　ASCII(美国信息交换标准代码)表

(American Standard Code for Information Interchange)

ASCII 码值	字符	ASCII 码值	字符
0	Null	31	cursor down
1	☺	32	Space
2	☻	33	!
3	♥	34	"
4	♦	35	#
5	♣	36	$
6	♠	37	%
7	Beep	38	&
8	◘	39	'
9	Tab	40	(
10	LineFeed	41	)
11	Cursor home	42	*
12	FormFeed	43	+
13	Carriage return	44	,
14	♫	45	-
15	☼	46	.
16	►	47	/
17	◄	48	0
18	↕	49	1
19	‼	50	2
20	¶	51	3
21	§	52	4
22	▬	53	5
23	↨	54	6
24	↑	55	7
25	↓	56	8
26	→	57	9
27	←	58	:
28	Cursor right	59	;
29	Cursor left	60	<
30	Cursor up	61	=

ASCII 码值	字符	ASCII 码值	字符
62	>	98	b
63	?	99	c
64	@	100	d
65	A	101	e
66	B	102	f
67	C	103	g
68	D	104	h
69	E	105	fi
70	F	106	j
71	G	107	k
72	H	108	l
73	I	109	m
74	J	110	n
75	K	111	o
76	L	112	p
77	M	113	q
78	N	114	r
79	O	115	s
80	P	116	t
81	Q	117	u
82	R	118	v
83	S	119	w
84	T	120	x
85	U	121	y
86	V	122	z
87	W	123	{
88	X	124	
89	Y	125	}
90	Z	126	-
91	[	127	⌂
92	\	128	Ç
93	]	129	ü
94	^	130	é
95	_	131	â
96	`	132	ä
97	a	133	à

ASCII 码值	字符	ASCII 码值	字符
134	å	170	¬
135	ç	171	½
136	ê	172	¼
137	ë	173	¡
138	è	174	«
139	ï	175	ƒ
140	î	176	╥
141	ì	177	╠
142	Ä	178	
143	Å	179	║
144	É	180	╟
145	æ	181	╞
146	Æ	182	╙
147	ô	183	╫
148	ö	184	╞
149	ò	185	├
150	û	186	│
151	ù	187	█
152	ÿ	188	▓
153	Ö	189	░
154	Ü	190	«
155	¢	191	┐
156	£	192	└
157	¥	193	┴
158	₧	194	┬
159	ƒ	195	├
160	á	196	─
161	í	197	┼
162	ó	198	╞
163	ú	199	╟
164	ñ	200	╚
165	Ñ	201	╔
166	ª	202	╩
167	º	203	╦
168	¿	204	╠
169	⌐	205	═

ASCII 码值	字符	ASCII 码值	字符
206	╬	231	τ
207	╧	232	Φ
208	╨	233	θ
209	╤	234	Ω
210	╥	235	δ
211	╙	236	∞
212	╘	237	φ
213	╒	238	ε
214	╓	239	∩
215	╫	240	≡
216	╪	241	±
217	┘	242	≥
218	┌	243	≤
219	█	244	⌠
220	▄	245	⌡
221	▌	246	÷
222	▐	247	≈
223	▀	248	°
224	α	249	∙
225	β	250	·
226	Γ	251	√
227	π	252	ⁿ
228	Σ	253	²
229	σ	254	■
230	μ	255	(blank FF)

附录 D Turbo Basic 编辑命令表

Turbo Basic 本身提供了一个全屏幕编辑器。Turbo Basic 编辑器提供了一系列编辑命令，它们是 WordStar 的子集，能方便地在编辑内容中查找或替换字符串；可以将任一字块存入磁盘或从磁盘中读入文件等。

为读者查阅方便起见，先将 Turbo Basic 的所有编辑命令分类列于表 D.1 中，然后再逐一详细介绍(表中"∧"表示 Ctrl 键)。

表 D.1 Turbo Basic 编辑命令表

作 用	命 令
移动光标的命令	
左移一个字符	∧S 或→
右移一个字符	∧D 或←
左移到词的开头	∧A 或∧→
右移到词的开头	∧F 或∧←
左移到行首	∧QS 或 Home
右移到行尾	∧QD 或 End
上移一行	∧E 或↑
下移一行	∧X 或↓
上移到显示窗口的第一行	∧QE 或∧Home
下移到显示窗口的最后一行	∧QX 或∧End
上移一屏	∧R 或 PgUp
下移一屏	∧C 或 PgUp
上移到程序的起始处	∧QR 或∧PgUp
下移到程序的末尾	∧QC 或∧PgOn
在程序中上卷一行	∧W
在程序中下卷一行	∧Z
移到字块的开始	∧QB
移到字块的末尾	∧QK
移到最后的光标位置	∧QP
设置查找标记 n (n=0—3)	∧Kn (不是∧n)
查找标记 n (n=0—3)	∧Qn (不是∧n)

删除命令	
删除光标左边的字符	∧H 或 Backspace
删除光标处的字符	∧G 或 Del
从光标位置开始删除到词尾	∧T
从光标位置开始删除到行尾	∧QY
删除整行	∧Y

字块操作命令	
标记字块的起点	∧KB 或 F7
标记段的终点	∧KK 或 F8
标记一个包含光标的词作为字块	∧KT
打印字块	∧KP
删去字块	∧KY
将字块复制到光标位置	∧KC
将字块移到光标位置	∧KV
将字块写入磁盘	∧KW
将磁盘文件读入光标处	∧KR
隐蔽／显示设了标记的字块	∧KH

模式转换	
打开或关闭自动缩排方式	∧QI
打开或关闭插入方式	∧V 或 Ins
打开或关闭制表方式	∧OT

查寻和替换命令	
查寻字符串	∧QF
查寻并替换字符串	∧QA
重复查寻或替换	∧L

存、取及退出命令	
把当前文件放入磁盘	∧KS 或 F2
从磁盘中读入新文件	F3
退出编辑回到主菜单	∧KD，∧KQ ∧KK 或 Esc

其它命令	
使操作无效	∧U
键入控制字符	∧P
移动行尾至新行	∧N
制表	∧I 或 Tab
恢复错误信息	∧QW
恢复一行	∧QL

一、移动光标命令

这一节叙述移动光标的 Turbo Basic 编辑命令

1. 左移一个字符（∧S或←）

这个命令使光标左移动一个字符，当光标在一行的起始位置时，Turbo BASIC不理睬这个命令。

2. 右移一个字符（∧D 或→）

这个命令使光标向右移动一个字符。这个命令不管作为这行结尾的最后的不空字符而继续向右移动光标直到它达到行的最大长度（248 个字符）。

3. 左移到词的开头（∧A 或←）

这个命令把光标移动到左边头一个词的起始位置。词定义为被以下17个符号：空格，$ ' （） · +，-。 / ； 〈〉 〔〕 ∧中的任何两个隔出的一串字符。如果光标在词中而未在词的开头，光标就移动到这个词的开头。如果不在词中，光标就移动到它左侧第一个词的开头。如果没有任何词位于光标的左侧并且在当前行上面还有一行，则光标就移动到前一行最后不空字符后面的位置。

4. 右移到词的开头（∧F 或∧→）

这个命令把光标移到右边第一个词的开头。如果光标在一行的最后一个词中，光标就移动到这个字符结尾后面的位置。如果光标在行尾并且另一行跟着当前行，光标就移动到下一行第一个词的起始位置。

5. 左移到行首（∧QS 或 Home）

这个命令把光标移动到当前行的第一列.

6. 右移到行尾（∧QD 或 End)

这个命令把光标移动到该行最后的不空字符后面的位置。某些情况下，这个命令会把光标向左移。

7. 上移一行（∧E 或↑）

如果有一行在当前行的上面，这个命令把光标向上移动到这一行．这个命令维持光标所在的列——它遇到一个制表字符时，光标会向右移.

8. 下移一行（∧X 或↓）

如果有一行在当前行的下面，这个命令把光标向下移动到这一行。这个命令维持光标所在列——除非它遇到一个制表字符，那样光标将进一步右移.

9. 上移到显示窗的第一行（∧QE 或 Home)

这个命令把光标移动到编辑窗内可以见到的第一行里，这个命令维持光标所在列——除非它遇到制表字符，那样光标将进一步右移.

10. 下移到显示窗的最后一行（∧QX 或∧End)

这个命令把光标移动到编辑窗内可以见到的最后一行里．这个命令维持光标所在的列——除非它遇到制表字符，那样光标将进一步右移.

11. 上移一屏（∧R 或 PgUp)

这个命令可在编辑窗内显示程序的前面的段落．一般地，窗内能见到的第一行变成为窗内能见到的一行.

12. 下移一屏（∧C 或 PgOn)

这个命令可在编辑窗内显示程序的后面的段落。一般地，窗内能见到的最后一行变成为窗内能见到的第最后一行.

13. 上移到程序的起始处（∧QR 或 PgUp)

这个命令把光标移动到程序第一行的起始位置，并把这行显示在编辑窗的最上部。

14. 下移到程序的末尾（∧QC 或 PgOn)。

这个命令把光标移动到程序最后一行的结尾，并把这行显示在编辑窗的底部。

15. 上卷一行（∧W）

如果编辑窗内显示出的第一行以上还有较多的程序行，那么这个命令把编辑窗中的每一行都向下移一行。以在编辑窗的顶部带入新的一行程序。这个命令维持光标所在的位置，除非光标在编辑窗能见到的最后一行上。

15. 下卷一行（∧Z）

如果编辑窗内显示出的最后一行下面还有较多的程序行，那么这个命令把编辑窗中的每一行都向上移一行以在编辑窗的底部带入新的一行程序。这个命令维持光标所在位置，除非光标在编辑窗能见到的第一行内。

16. 移到字块的起点（∧QB）

如果执行此命令之前已设置了文章的字块的起点标记，这个命令把光标移到这个字块起始标记的位置。

17. 移到字块的结尾（∧QK）

如果在执行这个命令之前已设置了文章的字块的结尾，这个命令把光标移到这个字块的结尾标记的位置。

18. 移动到最后的光标位置（∧QP）

这个命令把光标移动到发出最后的命令或键入最后的字符之前它所在的位置。这个命令在包括∧QF 在内的光标移动命令发出后，要使光标返回原操作位置是很有用的。

19. 设置查找标记 n（∧Kn）

这个命令在光标当前位置放置一个临时标记；n可以是0—3这四个个位数中的任何一个。（注：当按这数字键时不要按下 Ctrl 键）设置标记之后，利用带有适当的 n 值查找位置标记 n 的命令，就可以从程序中的任何位置立即返回。可以简单地通过在一处新的位置设立标记而移动位置标记。Turbo Basic 并不把位置标记与程序存放在一起。

20. 查找标记 n（∧Qn）

这个命令把光标移动到最后设置位置标记n的位置，n可以是0—3这四个个位数中的任何一个。

二、删除命令

这一节叙述删除文本的 Turbo BASIC 编辑程序的命令。

1. 删除光标左边的字符（∧H或Backspace）

这个命令立即删掉光标左方的字符。如果光标在行的起始位置，并且有一行处于当前行的上方，则这个命令把当前行连接到上行的结尾处。

2. 删除光标位置的字符（∧G 或 Del）

这个命令删除位于当前光标所在位置的字符。如果光标在一行的最后不空字符的后面并且在当前行的下面有一行。这个命令就把下一行连接到当前行的末尾。

3. 从光标删除到词尾（∧T）

这个命令删除从光标当前位置到光标所在的这个词的词尾的所有字符。如果光标处在一个空格上，那么这个命令删除从光标位置到后面邻近的不空字符之间的所有空格。如果光标在一行的最后不空字符的后面，并且在当前行的下面有一行，这个命令就把下面一行连接到当前行的末尾。

4. 从光标删除到行尾（∧QY）

这个命令删除从位于光标当前位置的字符到这一行结尾字符之间的一切字符。

5. 删除全行（∧Y）

这个命令删除光标所在行的所有字符，并且向上移动后继各行填补失缺。

三、字块操作命令

这节叙述对文章的字块起作用的 Turbo Basic 编辑程序的命令

1. 标记字块的起点（∧KB或F7）

这个命令标记位于光标当前位置的字符为一个文本字块的首字符。如果字块的结尾也被标记，这个命令就突出了字块中的文本。可以用一个简单的操作打印，移动，复制或删除被标记的文本字块，这些将在下面各节叙述。

2. 标记字块的末尾 （∧KK 或 F8）

这个命令标记位于光标当前位置的左边位置的字符为一个文本字块的尾字符，如果字块的起点也被标记，这个命令就突出了字块中的文章，可以用一个简单的操作打印，移动，复制或删除被标记的文本段，这些将在下面各节叙述。

3. 标记含有光标的词为字块（∧KT）

这个命令标记含有光标的词为一个字块．如果光标不在一个词的内部，这个命令就标记光标右边的第一个词．

4. 打印字块（∧KP）

这个命令把已被标明的字块中的文章打印在行印机上．如果没有标记字块或被标明的字块被藏蔽，那么这个命令就打印内存中的全程序。

5. 删除字块（∧KY）

这个命令从内存中删除当前程序里被标记的整个字块。

6. 把字块复制到光标位置（∧KC）

这个命令把当前程序中被标记的文本字块复制到光标的当前位置．在光标处及在其后的文本向后移动，为复制的字块留出余地．光标位于复制的字块的起点的左边．这个命令不影响原标记的文本字块．

7. 把字块移到光标位置（∧KV）

这个命令把当前程序中被标记的文本移动到光标的当前位置，并且从它的原来位置上删掉它．这个命令把在光标处及在其后的文本下推为字块的移动留出余地，光标位于移动后的字块的起点的左边．

8. 把字块写入磁盘（∧KW）

这个命令把当前程序中设了标记的文本字块复制到一个磁盘文件，并找一个文件名写在上面．如果已经存在这一文件，这命令就提出警告并且给以取消或给出新名字两种选择．这个命令不影响原设了标记的字块．

9. 将磁盘文件读入光标处（∧KR）

这个命令要求一个存在的磁盘文件名，然后把这一文件的内容插入到当前程序的光标的当前位置．这个命令把在光标处及在其后的文本下推，为从磁盘中插入的文本留下余地．这个命令自动地标记插入部分为字块，光标移在这个字块的起点的左边．这个命令对于从那里读出被插入文本的磁盘文件没有影响．

10. 隐蔽或显示设了标记的字块（∧KH）

当一个设了标记的文本字块正被显示时，这个命令可通过使这个字块不突出而隐蔽起设了标记字块存在的事实．一旦一个字块被隐蔽，就无法执行一个隐蔽字块的像字块复制，字块移动，字块写入等字块命令．如果一个设了标记字块存在但是是被隐蔽的，那么这个命令就重新突出地显示这字块并允许执行字块命令．

四、模式切换

这节叙述切换各种不同模式的 Turbo BASIC 编辑程序的命令。

1. 自动缩排方式的开关（∧OI）

这个命令把自动缩排方式的状态从关闭变为开启，或从开启变为关闭。当自动缩排方式开启时，只要按 ENTER 键产生一个新行，Turbo BASIC 就自动地使光标处于前一行第一个不空字符的同一列的位置。（注按 ENTER 键可产生一个新行，仅当插入方式开启。）

2. 插入方式的开关（∧V 或 Ins）

这个命令把插入方式的状态从关闭变为开启，或从开启变为关闭。当插入方式开启时，Turbo BASIC 把现存文本移到右侧，为进入的新文本留出余地。当插入方式关闭时，新进入的文本替换了现存文本。即把新文本写到现存文本的上面。注：仅当插入方式开启时才可把新的程序行加到一个程序的底部。

3. 制表模式的开关（∧OT）

这个命令把制表模式的状态从关闭变为开启，或从开启变为关闭。当制表模式开启时，可利用 TAB 在程序中设置跳格以使程序易读。Turbo BASIC 在第 9,17, 25，33，41 等列预置了制表停顿位。当制表方式关闭时，在 Turbo BASIC 中使 TAB 不产生作用。

五、查找与替换命令

这一节叙述查找与替换的 Turbo BASIC 命令。

1. 查找文本（∧QF）

这个命令要求一个由待查找的字符组成的字符串，要求一个查找选择项，并在程序中找到这个字符串。查找选择项可以是任意个数及下面的项目：

B　向前搜索该程序。

G　全局查找（搜索整个程序）而不仅是查找当前光标位置后（或前）的部分。

N　忽略查找字符串的前N—1次匹配，而把光标移到找到的第N个匹配上。例如，给出 4 作为查找选择项。使 Turbo BASIC 忽略带有查找字符串

的前 3 次匹配。而把光标移到第 4 次匹配上。

U 当检查查找字符串的匹配时，忽略字母大小写的差别。例如，给出U选择项，对 time 的查找和对 Time，TIME 等的查找等价。

W 仅把查找字符串和整个的单词匹配。当查找字符串是一个较长的词的一部分时，TurboBASIC 不作匹配。例如，若给出 W 选择项，查找 end 时将不与 ends，sends 或 friend 匹配。

查找选择的两个例子，一个是GB，对于查找字符串的匹配从程序的结尾一直查到开头。另一个是 3W，从当前光标位置向后搜索查找字符串和整个单词的第三次匹配。

2. 查找和替换文本（∧QA）

这个命令需要一个要查找的字符组成的字符串和一个由替换字符组成的字符串，需要查找和替换选择项，根据查找选择项在程序中查找字符串并且用替换字符串代替匹配项。查找选择项与对于查找文本命令所述一致。也可用下面的替换选择项：

N 无须先向用户核实找到的字符串是否要被替换，就完成替换动作。

3. 重复最近的查找或替换（∧L）

这个命令再次执行使用同一查找字符串替换字符串（如适当）与选择的最近一次查找或查找替换命令。

六、存贮，装入与退出命令

这一节叙述了有关存储，装入与退出的 Turbo BASIC 编辑程序的命令。

1. 把当前文件存入磁盘（∧KS或F2）

这个命令把当前在内存中的程序以程序名存入磁盘文件。如果程序名是 NONAME. BAS，这个命令要求给出另一个名字。如果一个同名磁盘文件业已存在，这个命令会询问是否要把新程序写到旧磁盘文件上。

2. 从磁盘上读新文件（F3）

这个命令查找要被装入内存的磁盘文件名，如果内存中没有当前程序，这个命令可使新程序装入内存。当这个命令把新文件装入内存时，当前在内存中的任何文本将会丢失。

3. 退出编辑回到主菜单（∧KD，∧KQ，∧KX 或 Esc）

这个命令停止编辑并且返加主菜单，编辑好的程序保持在内存以备进行编译，执行或贮存。注意：退出编辑程序并不自动地使 Turbo Basic 把程序存入磁盘。必须在编辑程序中按 F2 这样的命令，才可将程序存盘。

七、其它

1. 撤消操作（∧U）

这个命令会停止当前被送入或执行的命令。然后必须按ESC。这个命令对于一个全局查找和替换命令在未执行完以前就停止它是特别有用的。

2. 输入控制字符（∧P）

这个命令把下一个输入的字符视为控制字符，而不是一个标准字符。例如，编辑程序把按 CTRL—P，以及然后按 M 视为字符∧M（ASCII 字符 13）。当指定查找和替换字符串时，首先要用这个命令。这个命令允许该字符串包含控制字符，例如制表字符∧I，以及行尾字符∧M∧J。

3. 移动行尾到新行（∧N）

这个命令在当前行的下方产生一个新行，并且把位于光标位置及其右边的任何文本移到这一新行的开头。光标留在当前行的新结尾处。

4. 制表（∧I 或 Tab）

如果制表方式是开启的，这个命令把制表符号放入程序并且把光标移动到下一个制表停顿处。Turbo BASIC 在第 9，17，25，33，41 等列预置制表停顿处。

5. 恢复错误信息（∧QW）

这个命令把最后的编译错误信息重新显示在编辑窗口的顶部，并且把光标移回到程序文本中错误发生的位置。

6. 恢复一行（∧QL）

这个命令把当前行恢复到它的原来的形式，即把光标从另一行移到该行时该行所具有的形式。利用这个命令可取消对当前行刚刚作出的修改。

附录 E　Turbo Basic 错误信息

Turbo Basic 程序可能有三类错误: 编译错误、运行错误和逻辑错误. 编译错误是编译时发现的, 运行错误是程序运行时才暴露出来的. 多数编译错误是语法错误, 是由于符号不对、括号不配对等原因造成的; 运行时错误是在执行编译后的目标程序时出现的, 例如, 企图求一个负数的平方根等等. 逻辑错误是由于程序的逻辑关系设计不当引起的, 常表现为运行结果不对. 该类错误主要靠测试手段解决, 在此不作介绍.

可以在程序中为运行时错误设置陷井: 使用 ON ERROR 语句, 使得一旦出现错误, 就转到事先设计好的程序段进行处理, 处理后再继续运行程序.

作为一种调试技术, 可以用 ERROR 语句为错误处理子程序模拟产生运行时错误.

下面分别介绍运行时错误和编译错误(先给出错误码, 错误信息, 然后给出可能的原因).

运行时错误

2　Syntax error (语法错误)

企图把字符串读(READ) 到数值变量中就会造成这样的语法错误, 其它语法错误在编译时就发现了.

3　RETURN without GOSUB(没有与RETURN语句相匹配的GOSUB语句)

4　Out of data (数据不够)

READ语句要读的数据比DATA语句中给出的多.

5　Illegal function call (非法的功能调用)

把不合适的变元传递给某些语句或函数时会导致这个错误, 这类情况约有101种:

- 颜色号或屏幕方式号太大了.
- 无图形适配器却要执行图形语句或要用SCREEN语句设置该方式.
- 企图执行非法的数学操作, 如求负数的平方根.
- 在GET或PUT语句中的记录号太大了, 或者是使用了负数.
- 企图对顺序文件使用 WIDTH 语句.

6 Overflow (溢出)

计算的结果超出了数据类型所允许的范围。如: x% = 32767+1就会产生溢出的错误，因为 32768 超出了整数的范围。只有在主菜单的 Options 项中将 Overflow 置为 ON，才能发现整数溢出的错误，但浮点数的溢出错误总能被发现。

7 Out of memory (内存不够)

许多情况都能导致这个错误，包括说明了太大的数组.

9 Subscript out of range (下标超界)

使用的下标超出了DIM语句对该数组作的说明. 只有将Options 菜单中的 Bounds 置为 ON 之后才进行这项检测.

10 Duplicate definition (双重定义)

没有先删除动态数组就企图重新说明它。

11 Division by zero (用零作除数)

企图用零作除数或求零的负数次幂.

13 Type mismatch (类型不匹配)

在需要数值的地方使用了字符串,或在需要字符串的地方使用了数字。在 DRAW、PLAY 等语句中容易发生这种错误.

14 Out of string space (字符串空间用完了)

字符串空间只限于64K.

15 String too long (字符串太长)

字符串表达式生成的字符串超过32767个字节

19 No RESUME (没有RESUME 语句)

在错误处理程序段中遇到使程序结束的语句. 在错误处理程序段中应该由 RESUME 语句退出该程序段. 这个错误通常是在错误处理程序段中丢失了 RESUME 语句.

20 RESUME without error (没有发生错误却遇到了RESUME 语句)

应该是出现程序错误，才转去执行处理错误的程序段；在执行处理错误程序时，遇到 RESUME 语句再返回主程序. 为防止意外地"掉进"错误处理程序段,应在该程序段前放上 GOTO 之类的语句.

24 Device time—out (设备超时)
为通讯状态线路规定的等待时间的期限到了。可以为ClearToSend, CarrierDetect 和 DataSetReady 等状态线指定计算机等待的时间 (time—out)。设备超进可以停止招待程序，也可以重新安排通讯操作。

25 Device fault (设备故障)
硬件设备（例如由于打印机接口或通讯口）造成的错误.

27 Out of paper (打印纸没有了)
由打印机接口反映的"没有开印纸了"的信息造成的。也可能是没有开打印机或其它问题引起的。

50 Field overflow (字段溢出)
FIELD 语句中为各字段变量分配的宽度超出了OPEN语句为记录指定的宽度。

51 Internal error (内部错误)
Turbo Basic运行时系统出现故障。

52 Bad file number (文件号不正确)
文件操作语句中给出的文件号和OPEN语句中指定的不一致，或是使用的文件号超出了允许的范围。

53 File not found (文件没找到)
在指定的磁盘上未找到指定的文件。

54 Bad file mode (文件方式不对)
对文件进行的操作与打开它时指定的方式不符，例如企图对顺序文件进行GET操作。

55 File already open (文件已经打开了)
企图打开或删除一个已经打开的文件。

57 Device I/O error (设备输入/输出错误)
执行某语句时设备出了问题

58 File already exists (文件已经存在)
用NAME 语句起的新名字和磁盘上的某文件重名。

61 Disk full (磁盘满了)
指定的或默认的盘上没有足够的空间来实现某文件操作.

62 Input past end (输入超过了文件尾)
企图从已经到了文件末尾的文件中继续读取数据. 使用EOF函数可以避免这个错误. 企图从采用 OUTPUT 或 APPEND 方式打开的文件中读取数据, 也会导致这种错误.

63 Bad record number (坏的记录号)
企图在随机文件的GET语句或PUT语句中用负数或大于16777215的数做为记录号.

64 Bad file name (非法的文件名)
在FILES, KILL, 或NAME 语句中指定的文件名中有不合法的字符.

67 Too many files (文件太多)
这个错误可能是由于企图在根目录中建立太多的文件引起的, 也可能是由于使用了非法的文件名而影响了 DOS 建立文件的系统调用所造成的.

68 Device unavailable (设备不能用)
企图打一个不存在的设备文件; 例如, 在一个没有串行口的系统中打开 COM1:.

69 Communications buffer overflow (通讯缓冲区溢出)
企图往已经装满字符的通讯缓冲区中输入新字符造成的错误. 在程序中应该更经常地检查该缓冲区并使它变空, 或者提供更大的缓冲空间.

70 Permission denied (许可被否认了)
企图往有写保护的盘上存写东西.

71 Disk not ready (磁盘没准备好)
没放磁盘或没关驱动器的门.

72 Disk media error (磁盘介质错误)
软盘或硬盘控制板指出磁盘在一个或几个扇区有错误.

74 Rcname across disks (改名工作跨越了磁盘)
为文件改名字时, 新旧文件名应属于同一个磁盘.

75　Path / File access crror (路径或文件不对)

在可以指定路径名的语句中 (如OPEN、NAME、MKDIR等)，使用了错误的路径。例如企图打开一个子目录，或者是企图删除一个有内容的目录。

76　Path not found (路径没找到)

找不到在CHDIR，OPEN 等语句指出的路径。

202　Out of string temp space (临时字符串空间太大)

字符串表达式所需的临时字符串缓冲区太长。象MID $ (LEFT $ (RIGHT $ (……)))这样的表达式就会导致这种错误。可以把中间结果赋给临时字符串变量以便简化表达式。

203　Mismatched common variables (公共变量不匹配)

企图链接(CHIAN)的两个程序段中的COMMON语句不匹配。Turbo Basic对COMMON 语句中的类型和个数都要进行检查。

204　Mismatched program options(程序的选择项不匹配)

企图链接的两个程序段是用不同的选择项编译的 (不同的数学函数库，不同的数学处理方法等等)。

205　Mismatched program revisions (程序的版本不同)

企图链接的两个程序段是用Turbo Basic 的不同版本的编译器生成的。

206　Invalid program file (无效的程序文件)

企图运行或链接的程序不是由Turbo Basic 编译生成的。

242　String / array memory corrupt (字符串的内存区或数组内存区有错误)

字符串或数组的内存区被错误地重写了。造成这个错误的原因可能是汇编语言子程序使用不当，对数组的访问超出了范围，或者是 Turbo Basic 运行时系统有错误.

243　CHAIN / RUN from EXE file only (要链接/ 运行的只能是.EXE文件)

在Turbo Basic 的环境内，企图链接或运行其它程序段将出现这种错误。要链接或调用的程序段必须事先被编译为.EXE 文件，而且只能在 DOS 中链接或调用其它程序段。

编译错误

401 Expression too complex (表达式太复杂)
表达式中的运算符和操作数太多. 可拆成两个或几个来简化表达式.

402 Statement too complex (句子太复杂)
句子太复杂会使内部编译缓冲区溢出. 可以把它拆成两个或几个句子.

403 $IF nesting overflow (条件编译的嵌套太多)
条件编译结构只许嵌套16层.

404 $INCLUDE nesting overflow (嵌入文件(子文件)的嵌套太多)
不包括主程序时, 嵌入文件(include file)最多可以嵌套4层.

405 Block nesting overflow (块结构嵌套太多)
块结构最多嵌套255层.

406 Compiler out of memory (编译器可以使用的内存不足)
编译器用于符号、缓冲区等等的内存空间用完了. 可以把程序拆成主程序和其余的几个程序段, 再用 $INCLUDE 元语句往主程序中嵌入那几个程序段. 如果在编译时遇到这个错误, 也可以在 Options 子菜单中把编译改为往磁盘上生成.EXE 文件.

407 Program too large (程序太长)
程序中的语句超过了65530个

408 Segment exceeds 64K (超过了64K段长的限制)
一个段中的内容超过了64K. 可用 $SEGMENT元语句使超出的代码放到其它段.

409 Variables exceed 64K (变量超过了64K)
标量变量的全部空间限定为64K, 其中要放字符串的指针、整型变量、长整型变量、单精度型和双精度型变量. 克服这个错误的方法是除去那些不用的变量或者把程序拆成主程序和 CHAIN (链接) 程序.

410 ","expected (此处应该有个逗号)

411 ";"expected (此处应该有个分号)

412 “(”expected （此处应该有个左括号）

413 “)”expected （此处应该有个右括号）

414 “=”expected （此处应该有等号）

415 “—”expected （此处应该有连字符）

416 Statement expected （这里应该是一个语句）
某些字符不能用来标识一个语句、元语句或变量.

417 Label/line number expected （应该有标号/行号）
IF、GOTO、GOSUB或ON语句中需要标号或行号

418 Numeric expression requires relational operator （数字表达式需用关系运算符）
编译器发现在一个数值运算符的旁边是一个字符串型的操作数就会出现这个错误。

419 String expression requires string operand (字符串表达式需用字符串操作数)
编译器发现在字符串表达式中有: X$ = A$ + B 之类的东西就会指出这种错误.

420 Scalarvariable expected （应该是标量）
编译器需要一个标量变量作为函数的形式参数. 标量变量包括字符串型、整型、长整型、单精度型和双精度型变量. 简单说, 这里不能用数组作形式参数.

421 Array variable expected （应该是数组变量）
在DIM语句或图形的GET、PUT语句中需要数组变量（不是数组元素).

422 Numeric variable expected （应该是数字变量）
在INCR, DECR 或CALL ABSOLUTE 等的偏移地址中需用数值变量.

423 String variable expected （应该是字符串变量）
在FIELD, GET$, PUT$ 或LINE INPUT 语句中需要用字符串变量.

424 Variable expected （应该是变量）

在VARPTR，VARPTR $ 或VARSEG函数中需要一个变量.

425 Integer constant expected （应该是整型常数）
在为命名常量赋值的语句中或在条件编译语句 $ IF / $ ELSEIF中，需用整型常数.

426 Positive integer constant expected （应该是正整型常数）
用DIM STATIC 说明数组时，下标的界域需使用正整型常数．在 $ SOUND，$ COM 和 $ STACK 元语句中也需要使用正整型常数.

427 String constant expected （应该是字符串常数）
在 $ INCLUDE 元语句中的文件名要用字符串常量.

428 Numeric scalar variable expected （应该是数值标量）
此处应使用数值标量，不论是整型、长整型、单精度型还是双精度型．例如在FOR / NEXT 循环中。

429 Integer scalar variable expected （应该是整型标量）
在CALL ABSOLUTE 语句中要用整型标量变量

430 Integer array variable expected (此处应使用整型数组)
如在PALETTE 语句中

431 End of line expected （应结束程序行）
在元语句、标号或END SUB 等后面不能再有其它内容（注释除外）

432 AS expected （应该有AS）
在FIELD 语句或OPEN 语句中需要有保留字AS.

433 DEF FN expected （应该有DEF FN）
编译器发现有END DEF 或EXIT DEF语句，却没找到函数定义的开始语句．定义函数必须用 DEF FN 语句开始.

434 IF expected （应该有IF）
编译器发现有END IF 或EXIT IF语句，却没找到开始的IF语句.

435 DO LOOP expected （应该有DO LOOP）
有LOOP或EXIT LOOP语句，却没找到开始的DO语句.

436 SELECT expected (应该有SELECT)

定义SELECT CASE 语句丢了保留字SELECT，或者编译遇到END SELECT 或 EXIT SELECT 语句，却没发现开始的 SELECT CASE 语句，都会导臻这个错误。此外使用保留这 CASE 作变量名也会导臻这个错误。

437 CASE expected (应该有CASE)

定义SELECT CASE 语句时，丢了保留字CASE。使用SELECT 作变量名也会导致这个错误

438 FOR LOOP expected (应该有FOR 循环)

有NEXT 或EXIT FOR 语句，却没有相应的FOR 语句

439 SUB expected (应该有SUB)

有END SUB 或EXIT SUB 语句，却没有过程定义开始的SUB语句

440 END DEF expected (应该有END DEF)

函数没有用相应的END DEF 语句结束

441 END IF expected (应该有END IF)

IF 块没有用相应的END IF 语句结束

442 LOOP／WEND expected (应该有LOOP／WEND)

DO 或 WHILE 循环未用LOOP 或WEND 语句结束

443 END SELECT expected (应该有END SELECT)

SELECT CASE 语句未用END SELECT 语句结束

444 END SUB expected (应该有END SUB)

过程没有用END SUB 语句结束

445 NEXT expected (应该有NEXT)

FOR 循环没有用NEXT 结束

446 THEN expected (应该有THEN)

IF 语句中丢掉了THEN 部分

447 TO expected (应该有TO)

FOR 语句中丢了TO部分

448 GOSUB expected (应该有GOSUB)
ON语句中丢了GOSUB 部分

449 GOTO expected (应该有GOTO)
ON 语句中丢了GOTO部分

450 $ ENDIF expected (应该有 $ ENDIF)
条件编译元语句 $ IF丢了其相应的 $ ENDIF

451 Unmatched $ ELSE (未匹配的 $ ELSE)
编译器遇到了 $ ELSE 元语句，前面却没有相应的 $ IF元语句

452 Unmatched $ ENDIF (未匹配的 $ ENDIF)
编译器遇到 $ ENDIF元语句，前面却没有相应的 $ IF元语句

453 Undefined name constant (使用了未定义的命名常量)
应对其进行定义或直接用它的值作常数

454 Undefined function reference (引用了未定义的函数)
在表达式中使用了一个未经定义的函数名．查看一下是否把函数名字弄错了，或者对函数进行定义．

455 Undefined subprogram reference (调用了未定义的过程)
用CALL 语句调用一个过程却未对其进行定义．看看是否把过程名弄错了，或者对该过程进行定义．

456 Undefined label / line reference (引用了未定义的标号或行号)
在IF，GOTO，GOSUB 或ON 语句中使用了标号或行号，却没有定义该标号或行号．查看是否把标号或行号弄错了，或者定义该标号和行号．

457 Undefined array reference (引用了未定义的数组)
使用了未经DIM 语句说明的数组．查看是否把数组名弄错了，或用DIM语句定义它．

458 Duplicate label / line number (有重名的标号或行号)
同样的标号或行号定义了两次．检查程序和内嵌文件中是否有相同的标号或行号，对它们进行修改．

459 Duplicate named constant (有重名的命名常量)

即一个命名常量被定义了两次.

460 Duplicate function definition (有重名的函数定义)
对同一函数名定义了两次

461 Duplicate subprogram definition (有重名的子程序)
用同一个子程序名定义了两个子程序.

462 Duplicate common variable (有重名的COMMAND变量)
在一个COMMON语句中有重名的变量.

463 Duplicate variable declaration (有同名的变量)
在LOCAL, STATIC, 或SHARED语句中有同名的变量.

464 Duplicate $ COM defintion (有重复定义的 $ COM元语句)
在程序中为每个通讯口只能使用一次 $ COM元语句.

465 Duplicate $ SOUND definition (用了两次 $ SOUND元语句)
程序中只能使用一次 $ SOUND元语句.

466 Duplicate $ STACK definition (用了两次 $ STACK 元语句)
程序中只能使用一个 $ STACK元语句.

467 Invalid line number (无效的行号)
行号必须在0到65535之间.

469 Meta—statements not allowed here (这里不许使用元语句)
元语句必须放在一行的开始.

470 Block / Scanned statements not allowed here 这里不能出现块结构语句
WHILE / WEND, DO / LOOP 和SELECT CASE 等块结构语句不许出现在单行的 IF 语句中.

471 Unknown identifier / syntax error (不认识的标识符 / 语法错误)
出现了编译器所不能识别的东西。

472 Array subscript error (数组下标不对)
使用的下标和说明该数组时的规定不符

473 Array bounds error (数组下标范围错误)

在引用一个静态数组时使用了超出其下标范围的值

474 Type mismatch (类型不匹配)
SWAP 语句中的变量应该是同一类型的。

475 Parameter mismatch (参数失配)
参数的类型或个数与在函数或过程定义中说明的不符

476 CLEAR parameter error—use MEMSET / $ STACK (CLEAR语句中带有参数)
在Turbo Basic 中，CLEAR 语句不允许带参数，这与PC BASIC不同：

477 LOCAL requires DEF FN / SUB (LOCAL 要求有DEF FN / SUB)
只能在函数或过程内用LOCAL 语句说明变量。

478 SHARED requires DEF FN / SUB (SHARED要求有DEF FN / SUB)
只能在函数或过程内用SHARED 语句说明变量。

479 STATIC rquires DEF FN / SUB (STATIC要求有DEF FN / SUB)
只能在函数或过程内用STATIC 语句说明变量

480 COMMON arrays must be dynamic
COMMON 语句中的数组必须是动态的。

481 LOCAL arrays must be dynamic (用LOCAL语句说明的数组必须是动态的)
改正这个错误的方法是去掉保留字STATIC，或者把数组的定义从函数或过程内移到主程序中作为全局的静态数组。

482 Parameter arrays cannot ~~be altered~~ (数组参数不能被改变)
在过程中不能对作为参数定义的数组使用DIM或ERASE语句.

483 Array is not STATIC (数组不是静态的)
用DIM 说明静态数组时，不能用变量说明其下标

484 Array previously STATIC (数组已经是静态的)
前面已经把该数组定义为静态的，就不能再对它执行适用于动态数组的操作，例如对同一数组进行两次说明 (DIM). 同样，已说明为动态的数组也不能再说明为静态的。

485 Array name without "( )" expected (数组名无"()")
删除数组时，数组名后不能带圆括号

486 Array exceeds 64K (数组大小超过64K)
一个数组的大小不能超过64K，如果需使用更大的数组，要把它拆成几个.

487 Arrays limited to eight dimensions (一个数组最多可有八维)
数组维数超过八维.

488 Invalid numeric format (无效的数字格式)
程序中用的数字超过了36位，或者是一个浮点数中使用了E却没给出指数值.

489 Invalid function / procedure name (无效的函数名或过程名)
函数名应在FN后先有一个字母，其后可以有字母、数字或圆点，最后可以有类型说明符（%，&，!，#或$）. 过程名应由字母开头，后面可有字母、数字或圆点，不能有类型说明符.

490 No parameters with INLINE SUB (INLINE SUB) 未带有形式参数)
在内部过程中没有指定形式参数表，而该汇编语言子程序却需要传送给它某些参数，编译器不进行类型检查.

491 Only $INLINE allowed here (此处只允许使用$INLINE)
内部过程 (INLINE SUB) 中只能使用$INLINE语句

492 $INLINE requires SUB INLINE (INLINE 要求SUB INLINE)
SINLINE 语句只能出现在内部过程中

493 $INCLUDE file not found (未找到INCLUDE 文件)
在指定的目录、现行的目录或由Setup 菜单设置的目录中没有找到所要的内嵌文件.

494 $INCLUDE disk error (内嵌文件时出现磁盘错误)
编译期间要打开内嵌文件时出现了磁盘错误.

495 $INCLUDE file too large (内嵌文件太长)
和Turbo Basic编辑器所能处理的所有文件一样，内嵌文件也不能大于64K.

496 $INLINE file not found (没找到指定的INLINE 文件)

497 $INLINE disk read error (INLINE文件磁盘读写错误)
编译期间企图打开INLINE 文件时，发现了磁盘错误.

498 Temporary file access error （临时文件访问错误)
在往磁盘上进行编译时 (生成.EXE 或.TBC文件)，编译器无法对临时文件进行访问. 检查在主菜单的 Setup 项中对各个目录的设置是否正确，指定的目录是否存在.

499 Destination file write error （目标文件存写错误)
在磁盘上进行编译时，编译器得到了磁盘存写工作有错误的信息.

600～610 Internal Error (Turbo Basic 系统内部的错误)

附录 F　Turbo Basic 指令集

本附录提供 Turbo Basic 的所有指令，并进行分类描述。为读者查阅的方便，本附录先简要给出各类指令的功能，然后再逐一分析各指令的功能、用法。

F.1　过程和函数一览表

链　接	
CHAIN	调用 Turbo Basic 的链接模块（扩展名为.TBC 或.EXE 的文件）
COMMAND	说明要传递给被链接程序的变量
RUN	重新运行程序

数 据 编 译	
DIM	定义数组
LET	为变量赋值
LOCAL	在过程或函数内说明局部变量
OPTION BASE	设定数组下标的下界
REM	允许重新读取 DATA 语句中的常数
SHARED	在过程或函数中定义全局变量
STATIC	在函数或过程中定义静态变量
TRON / TROFF	控制跟踪状态

设　备	
COM (n)	控制串行通讯口的陷井是否有效
INP	从 I / O 端口读入一个字节
IOCTL	与设备驱动器通讯
OPEN COM	打开并配置一个通讯端口。
OUT	将一字节输出到 I / O 端口

DOS	
CHDIR	改变当前目录（相当于 DOS 的 CHDIR 或 CD 命令）
KILL	删除文件（类似于 DOS 中的 DEL 命令）
MKDIR	创建一个子目录（类似于 DOS 的 MD 命令）
RMDIR	将字符型数据移入随机文件缓冲区.
SHELL	装入并执行子过程
SYSTEM	中止程序

错 误 处 理	
ERADR	返回最近发生错误处的地址
ERDEV	返回设备错误的类型
ERRDEV$	返回发生错误的设备名称
ERL	返回最近发生错误的行号
ERR	返回最近发生错误的错误号
ERROR	模拟运行时错误

文 件	
BLOAD	将用 BSAVE 语句存贮的文件装入内存
BSAVE	将某一内存区域的内容存入磁盘
CLOSE	关闭文件或设备
EOF	返回文件结束状态
FIELD	定义随机文件缓冲区的字段变量
FILES	显示目录内容（类似于 DOS 命令中的 DIR 命令）
GET(文件)	从随机文件中读取一条记录
GET(图形)	将图形屏幕的整体或部分存入数组
LINE INPUT#	从顺序文件中将一行数据读入一个字符串变量；忽略定界符
LOC	返回文件指针的位置（当前位置）
LOF	返回磁盘文件的长度或通讯文件的通讯缓冲区中剩余可用空间的大小。

LSET	将字符串数据放入随机文件缓冲区.
NAME	为文件改名（类似于 DOS 的 REN 命令）
OPEN	为读／写操作准备一个文件或一台设备
PRINT#	将数据写入顺序文件
PUT	往随机文件中写一条记录
PUT $	将字符串写入二进制文件中的当前位置
RESET	关闭并清除所有的文件
RSET	将字符型数据移入随机文件缓冲区
SEEK	在二进制文件中为 GET $ 和 PUT $ 定位
WRITE#	向顺序文件中输出数据

流 程 控 制

CALL	调用一个过程（子程序）
CALL ABSOLUTE	调用汇编语言子程序
CALL INTERUPT	调用系统中断
DEF FN／END DEF	自定义函数
DO／LOOP	在其顶部或底部带有条件测试的循环结构
END	中止程序或定义一个结构块的末尾
EXIT	提前退出一个结构
FOR／NEXT	定义一个自动增值（或减值）的循环
GOSUB	调用子程序
GOTO	使程序的流程转到标号处.
IF	测试条件，并在条件满足时改变程序流程
IF(块)	创建一系列 IF 语句
RESUME	在出错处理程序段中指定返回地址.
RETURN	从子程序中返回调用程序.
SELECT CASE	在过程或函数中定义全局变量
STOP	结束程序
SUB／END SUB	定义一个过程（子程序）
SUB／ INLINE	定义一个内部(机器码)过程
WHILE／WEND	建立一个在其顶部进行条件测试的循环

图 形

CIRCLE	画园或园的一部分
DRAW	在图形屏幕上画图

LINE	画直线、方框或方块
PAINT	在图形屏幕上用选定的颜色填充封闭区域
PEEK	返回指定存储单元的值（一个字节的值）
PEN (函数)	读取光笔状态
PEN (语句)	控制光笔事件的检查
PMAP	图形屏幕的物理坐标与实用坐标的相互转换
POINT	返回某象素的颜色／获取前一个点（LPR）的坐标
POKE	往指定地址写一个字节
PRESET	在图形屏幕上画一点
PSET	在图形屏幕上画一个点
PUT	往随机文件中写一条记录
VIEW	在图形屏幕上定义一个工作区域（视口）
WINDOW	定义图形坐标系统

硬 件 事 件

ON COM (n)	设置通讯陷井
ON ERROR	设置错误陷井
ON / GOSUB	根据数值表达式的值调用几个可能的子程序中的一个
ON / GOTO	根据数值表达式的值，将程序流程转到几个可能的程序段中的一个.
ON KEY(n)	若特定的键被按，则执行陷井捕获子程序
ON PEN GOSUB	如果发生光笔活动，语句便执行陷井捕获子程序.
ON PLAY	如果背景音乐缓冲区包含的音符数目少于指定数目，ON PLAY 语句便执行陷井捕获子程序.
ON STRIG	设置游戏棒陷井
ON TIMER	每隔 n 秒便执行陷井捕获子程序一次.
STACK	返回游戏棒的位置
STRIG(函数)	返回游戏棒的按钮状态.
STRIG (语句)	控制游戏棒事件的检查
WAIT	暂停程序，等待从端口 port 输入满足条件的位图

输　入	
INKEY $	从键盘上读取字符串，但不显示。
INPUT	提示用户为变量赋值
INPUT #	从顺序文件中将数据读入变量
INPUT $	从键盘或文件中读入指定数目的字符
INSTAT	读取键盘状态
LINE INPUT	从键盘上把一行数据赋给一个字符串变量，忽略定界符。
READ	把 DATA 语句中的常数读给变量
RESTORE	允许重新读取 DATA 语句中的常数。

键盘处理	
KEY	设定显示功能键的内容，或者定义击键陷井值。
KEY (n)	开/关某特定键的击键陷井

内存管理	
CLEAR	清除变量空间
DEF SEG	为 BLOAD, BSAVE, CALL ABSOLUTE, PEEK 和 POKE 语句定义数据段。
ENDMEM	返回物理存储器结束处的地址
ERASE	删除动态数组，初始化静态数组。
FRE	返回可供程序使用的剩余内存空间大小。
MEMSET	规定内存的上限
POKE	往指定的内存地址中写一个字节
VARPTR	返回变量的偏移地址
VARPTR $	返回变量地址的字符串形式
VARSEG	返回变量的段地址

原 语 句

$COM	为串行通讯的接收缓冲区分配内存
$DYNAMIC	设定数组的内存分配形式为动态的
$EVENT	控制事件捕获的代码生成
$IF／$ELSE／$ENDIF	根据给定的条件，有选择地编译程序块（即条件编译）
$INCLUDE	插入一个源程序文件（子文件）
$INLINE	在内部过程中插入机器代码
$SEGMENT	定义一个新的代码段
$SOUND	定义背景音乐缓冲区的大小
$STACK	定义运行时堆栈的大小
$STATIC	设定数组的内存分配形式为静态的.

杂 类 操 作

DATA	为 READ 语句准备数据
DELAY	暂停一段时间
REG	读取寄存器缓冲区中的数值，或为寄存器赋值
SWAP	交换两个变量的值

数 值 运 算

ABS	求绝对值
ASC	求字符串中第一个字符的 ASCII 码值
ATN	求反正切值
CDBL	将数据转换为双精度浮点数.
CEIL	返回一个大于或等于其自变量的最小整数.
CINT	对数据进行舍入取整
CLNG	将数值转换为长整型数
COS	返回自变量的余弦值
CSNG	将数值转换为等价的单精度值
CVI	将从随机文件中读取的字符串型数据转换为数值型.
CVMD	将从 Microsoft 格式的随机文件中读取的字符串型数据转换为 IEEE 格式的浮点数

DECR	设置或获取系统日期
DEFINT / DEFLNG / DEFSNG / DEFDBL / DEFSTR	定义变量的缺省类型
EXP	指数运算：e^x
EXP2	指数运算：2^x
EXP10	指数运算：10^x
FIX	截去小数部分，取整.
INCR	增大变量
INT	将数值取整
LOG	计算自然对数(以 e 为底)
LOG2	计算对数(以 2 为底)
LOG10	计算对数(以 10 为底)
MKI$ / MKL$ / MKS$ / MKD$	将数据转换为要输入到随机文件中的字符串
MKMS$ / MKMD$	将数据转换成 Microsoft 格式的字符串（用于随机文件的输出）
RANDOMIZE	随机数产生器的种子
RND	产生随机数
SGN	符号（判断）函数
SIN	返回正弦值
SQR	返回数值的平方根
TAN	返回正切值

输 出

PRINT	将数据送到屏幕输出
PRINT USING	将格式化信息送到屏幕输出
PRINT SPC	跳过若干个空格
TAB	在打印时指定数据输入出位置.
WRITE	往屏幕上发送带逗号定界符的数据

打 印 机 操 作

LPOS	获取打印缓冲区的当前位置
LPRINT	向打印机（LPT1:）发送数据
LPRINT USING	向打印机（LPT1:）发送格式化数据

屏幕操作	
CLS	清屏
COLOR(图形)	为图形操作设定颜色
COLOR(文本)	设定文本颜色
CSRLIN	返回光标的纵向位置（行号）
LOCATE	给光标定位和／或定义光标的形状
POS	返回当前光标的横向位置（列数）.
SCREEN (文件)	返回屏幕上指定字符的 ASCII 码或字符属性
SCREEN(图形)	设置屏幕模式
WIDTH	设置输出的行宽

声音	
PLAY (函数)	得到背景音乐缓冲区中音符的个数
PLAY(语句)	控制音乐陷井的状态
SOUND	产生指定频率和时间的声音

字符串操作	
BIN $	返回一个数值的二进制形式字符串
CHR $	返回 ASCII 码对应的字符
COMMAND $	在 DOS 状态下启动程序时，获取命令行中的其它信息。
GET $	从二进制文件中读取数据
HEX $	将十进制数转换为相应的十六进制格式字符串
INSTR	字符串查找
LCASE $	将字符串中的字母全部变为小写
LEFT $	获取字符串最左边的 n 个字符
LEN $	求字符串的长度
MID $	替换字符串／自字符串中提取子字符串
OCT $	返回数值的八进制形式字符串
RIGHT $	从字符串的右边开始提起子字符串

SPACE $	返回指定数目的空格字符串
STR $	将数值转化为字符串
STRING $	返加若干个相同字符给成的字符串.
TIMER $	设置系统时间
TIMER $	返回系统时间
UCASE $	将字符串中字母全部变成大写字母
VAL	将字符串转换为数值

F.2 Turbo Basic 命令的语法格式及其简述

Turbo Basic 中的语句分为两类：普通语句和原语句（Metastatement）。前者在程序运行过程中进行操作，发生作用；而后者则在编译时发生作用，控制编译器的工作方式，对程序进行特殊处理。Turbo Basic 中所有原语句均以 $ 打头，书写时必须单独占一行(后面可跟注释)，不能放在注释之中。

为查阅方便，本节先介绍原语句，然后介绍普通语句；各语句的排列顺序以字母顺序为准。

为帮助读者对语句格式的理解，下面给出文中用到的几个格式符号的约定：

[ ] —— 表示其中的内容是任选项

… —— 表示它前面的条目可以多次重复.

{ } —— 表示其中含有多个条目；使用时，必须选用，也只能选用其中用“ | ”分隔的一个条目。

| —— 用于分隔各条目，表示被分隔的条目不同时出现在语句中.

／ —— 表示它前后的元素为一个语句体或结构；使用时，／处可插入相应的语句组.

$ COM

功能：为串行通讯的接收缓冲区分配内存

格式：$ COMn size

注释：n表示通讯口号；size 为缓冲器的大小，缺省值为256（字节）. 在程序中使用此语句可取代系统缺省值或 Options 选单中的设定值。

$ DYNAMIC

功能：设定数组的内存分配形式为动态的

格式：$ DYNAMIC

注释:　动态数组可用ERASE命令消除，空出不再使用的数组所占用的内存。系统缺省设置为静态数组。

使数组自动分配成动态内存的方法有三个:

(1)　在DIM语句中使用关键词DYNAMIC;
(2)　用LOCAL表示数组对一个函数或过程是局部的;
(3)　用变量为数组确定维数，或在多个DIM语句中使用同一数组（不包括在过程或函数中作的LOCAL说明)。

在DIM语句中，可插入关键词STATIC控制$DYNAMIC原语句，使说明的数组为静态的。

$EVENT

功能:　控制事件捕获的代码生成

格式:　$EVENT {ON　　DFF}

注释:　$ EVENT ON 命令编译器编译源程序中的续行时插入附加代码，而$ EVENTOFF 则令编译器不插入附加代码，这对于允许事件捕获是必须的。

$IF / $ELSE / $ENDIF

功能:　根据给定的条件，有选择地编译程序块（即条件编译)

格式:

```
IF 条件
     语句组
[$ELSE
     语句组]
$ENDIF
```

注释:　Const是一个常数（~~逻辑值~~；它的真假控制着要编译的语句组。即Const为非零时，编译$IF至$ELSE间的语句；当Const为零时，编译$ELSE后面的语句。

$INCLUDE

功能:　插入一个源程序文件（子文件)

格式:　$INCLUDE filespec

注释:　filespec为文件名。$INCLUDE原语句命令编译器编译指定的磁盘文件，好像该文件出现在$INCLUDE语句所在处一样。

$ INLINE

功能:　在内部过程中插入机器代码

格式:　$ INLINE [filespec] |[byte1, byte2, …]

注释:　filespec 是包含机器语言代码的文件名; bytel, byte2 等为机器语言代码的实际字节 (值域为: 0～FFH).

$ SEGMENT

功能:　定义一个新的代码段.

格式:　$ SEGMENT

注释:　一个机器代码段最长可达64K字节. 如果一个程序很大, 以致于编译后的机器代码超出 64K 字节, 则必须在程序中放入原语句 $ SEGMENT 通知计算机启用下一个代码段 (Turbo Basic 允许程序的机器代码最多使用 16 个代码段).

注意:　不能在结构块中使用 $ SEGMENT 原语句.

$ SOUND

功能:　定义背景音乐缓冲区的大小

格式:　$ SOUND n

注释:　n为数值常数; 用于指定音乐缓冲区的大小. 每个音符占8个字节, 最多可有4096个音符, 所以缓冲区最大可为 32K. 缺省的容量为 32 个音符, 即 256 个字节. n 按音符数给定.

$ STACK

功能:　定义运行时堆栈的大小

格式:　$ STACK count

注释:　Count 是1024到32768之间的常数或命名变量. 缺省值为1024个字节, 缺省值可以被程序中插入的 $ STACK 原语句取代. 可用 FRE (—2) 函数查看剩余的堆栈空间大小.

$ STATIC

功能:　设定数组的内存分配形式为静态的

格式:　$ STATIC

注释:　$ STATIC原语句使编译器对其后出现在DIM 语句中的数组进行静态内存分配这种方式为系统缺省方式. 当下列情况出现时, 静态分配不能进行:

(1)　DIM语句中含有关键词DYNAMIC;

(2)　用LOCAL宣布数组对一个过程或函数是局部数组时;

(3) 用一个变量对数组定维;

(4) 数组在多个DIM语句中出现(不包括在过程或函数中作的LOCAL说明)。

ABS

功能: 求绝对值

格式: ABS (X)

注释: X可以是具体的数值,也可以是表达式。

ASC

功能: 求字符串中第一个字符的ASCII码值.

格式: ASC (X $)

注释: ASC函数返回一个0~255的整数; X $ 为字符串.

ATN

功能: 求反正切值

格式: ATN (X)

注释: X可以是具体的数值,也可以是变量、表达式; 单位必须为弧度.

BEEP

功能: 使扬声器发声

格式: BEEP [n]

注释: 发生一个频率为800Hz,持续时间为0.25秒的声音. n为任选项,表示连续发声的次数.

BIN $

功能: 返回一个数值的二进制形式字符串

格式: BIN $ (X)

注释: X应为-65535~65535~之间的整数. 若X为负数,则给出的其补码的字符串; X也可以是表达式. BIN $ 函数对 X 自动取整,小数部分四舍五入。

BLOAD

功能: 将用BSAVE 语句存贮的文件装入内存

格式: BLOAD filespec [,address]

注释: filespec是指明要装入文件并符合标准DOS命名约定的字符串表达式，可包括驱动器说明符和／或路径. address是一个结果在0～65535间的数值表达式，表示偏移地址，指明要从当前段中什么位置开始装入文件. 若省略了address，则在用BSAVE存盘前的地方装入文件.

注意: 不能直接装入EGA存储器. 由于对地址没有检测，所以BLOAD可能会搞乱系统.

BSAVE

功能: 将某一内存区域的内容存入磁盘

格式: BSAVE filespec, address, length

注释: filespec是字符串表达式，指明数据将存入的文件. filespec必须符合标准的DOS命名约定，也可包括驱动器说明符和／或路径信息. address是数值范围在0～65535之间的数值表达式，表示filespec开始存放的偏移地址（参见DEF SEG语句）. length是存储的字节数，其范围为0～65535.

CALL

功能: 调用一个过程（子程序）

格式: CALL procname [(parameter list)]

注释: procname是在程序中其它地方用SUB语句定义的过程名. parameter list(参数表）是任选项，表示将传递到procname中的变量表、表达式表和／或常数表；参数表中各参数间用逗号分隔.

CALL ABSOLUTE

功能: 调用汇编语言子程序

格式: CALL ABSOLUTE address(parameter list)

注释: address是包含将调用外部程序偏移地址的数字标量，段地址由最后使用的DEF SEG语句定义. address的范围为: —32768～65535，负数将由编译器转换为正数. Parameter list(参数表）表示要传递的参数；各参数用逗号分隔.

CALL INTERRUPT

功能: 调用系统中断

格式: CALL INTERRUPT n

注释: n是整数表达式，表示要调用的中断号；其数值范围为0～255. 在中断处理程序接受控制之前，处理器的寄存器要装入机器寄存器缓冲区中的数值. 中断处理程序返回时，寄存器缓冲区保存处理器寄存器中的数值. 在任意给定时刻，寄存器缓冲区包含处理器寄存器在最后的外部调用或REG语句完成时的状态.

CDBL

功能:　将数据转换为双精度浮点数

格式:　CDBL (n)

注释:　n为任意的整数、长整数或单精度浮点数; n也可以是相应的数值表达式. CDBL函数的返回值为双精度浮点型.

CEIL

功能:　返回一个大于或等于其自变量的最小整数.

格式:　CEIL (X)

注释:　X为任意的数值表达式; 函数返回值为整型.

CHAIN

功能:　调用Turbo Basic的链接模块 (扩展名为.TBC或.EXE的文件)

格式:　CHAIN filespec

注释:　filespec是一个符合标准DOS命名约定, 表示Turbo Basic链接模块 (扩展名为.TBC) 的字符串表达式. 如果省略扩展名, 设定为.TBC. 因此, 若要指定.EXE文件, 必须将.EXE附加到文件名上. 当前程序和目标程序中的COMMAND语句将其中的变量由当前程序传递到目标程序.

注意:　当前程序和目标程序两者中的COMMAND语句可含有不同的参数名, 介参数个数和对应的参数类型必须相同; 否则, 将产生运行时错误.

CHDIR

功能:　改变当前目录 (相当于DOS的CHDIR或CD命令)

格式:　CHDIR Path

注释:　Path(路径) 是一个符合DOS路径约定的字符串表达式. 如果path为无效的目录, 则发生76号运行时错误: Path Not found.

CHR $

功能:　返回ASCII码对应的字符

格式:　CHR $ (X)

注释:　X为0～255的整数或数值表达式. 当X为浮点数时, CHR $函数自动将其四舍五入取整. 函数返回值为字符串型.

CINT

功能: 对数据进行舍入取整

格式: CINT(X)

注释: X为数值表达式. CINT 函数不总是四舍五入; 当小数部分为0.5时, 向最近的偶数舍入. 函数返回值为整型.

CIRCLE

功能: 画园或园的一部分

格式: CIRCLE(X,Y), radius [, color [, start, end [, aspect]]]

注释: (x,y) 为园心坐标既可用绝对值形式, 也可用相对值形式 (使用STEP) 指定.
radius(半径) 为数值表达式, 控制园的大小.
Color(颜色) 是整数表达式, 确定园的颜色. 在中分辩图形模式下, 可用值为0~3 (缺省值为 3); 在高分辩率图形模式下, 可用值为 0 或 1 (缺省值为 1). 缺省的颜色为当前模式下最高的可用颜色值. 使用超出这些范围的颜色值时, 将产生第 5 号运行时错误: Illegal Function Call (有关颜色选择的详情参见 COLOR 语句).
Start(起始点) 和end(结束点) 为任选参数, 确定要画出扇形的起始和结束点, 按弧度计. 只想画园的一部分时, 使用这两个参数. 如果省 Start 和 end, 则画整个园.
aspect(园度) 确定园度 (或叫椭园度), 表示椭园或园的横轴与纵轴之比.

CLEAR

功能: 清除变量空间

格式: CLEAR

注释: CLEAR 是一个无参数命令, 将数字变量置0, 字符变量设定为空. CLEAR 语句关闭任何打开的事件捕获 (如正在演奏的音乐), 并重新初始化数组和字符串内存.

注意: 循环结构中出现CLEAR 时, 将清除计数器, 导致死循环.

CLNG

功能: 将数值转换为长整型数

格式: CLNG (X)

注释: X为数值表达式; 若X的小数部分为0.5时, 舍入近的偶整数. X 的数值范围为: $-2^{31} \sim 2^{31}-1$. 返回值为长整型.

CLOSE

功能: 关闭文件或设备

格式: CLOSE [[#]filenum [,[#]filenum]···]

注释: CLOSE 语句中止在OPEN语句中建立的文件号和文件的关系，并结束对该文件的输入/输出，文件缓冲区被清除，且当 filenum(文件号) 代表一个文件（而不是一个设备）时，DOS 中的 CLOSE 命令作用于该文件上，调整目录。如果 filenum 被省略，CLOSE 将关闭所有打开的文件。

CLS

功能: 清屏

格式: CLS

注释: 在文本模式下，CLS语句将屏幕清为背景色，并使光标回到屏幕左上角（第一行，第一列）；在图形模式下，它将屏幕也清为背景色，但 LPR（参数点）处于屏幕的中心，如果已执行过 VIEW 语句，则 CLS 只清视口；要想清全屏幕，则必须先用无参数的 VIEW 语句关掉视口，再执行 CLS 语句。

COLOR(图形模式)

功能: 为图形操作设定颜色

格式: 对于SCREEN 1:

COLOR [background] [, [palett]]

对于SCREEN 7, 8, 9 和10:

COLOR [foreground][, [background]]

注释: foreground(前景色) 是一个其值为0～15的整数表达式。

background(背景色) 是一个数值为0～63的整数表达式，指定背景颜色。

palette(调色板) 是一个数值为0或1的整数表达式，选择两个颜色调色板中的一个。

COLOR(文本模式)

功能: 设定文本颜色

格式: COLOR [foreground][, [background][,border]]

注释: foreground(前景色) 是一个0～31整数表达式，指定字符颜色。

background(背景色) 是一个数值为0～7的整数表达式，指定背景颜色。

border(边框颜色) 是一个数值为0～15的整数表达式，指定边框颜色。

如果这些参数中任意一个超出指定的数值范围，将发生第5号运行时错误:

Illegal Function Call(非法的功能调用)

COM

功能: 控制串行通讯口的陷井是否有效

格式: COM (n) {ON|OFF|STOP}

注释: n为通信端口号 (1或2). ON, OFF和STOP分别表示通讯陷井有效、无效和暂停使用.

COMMAND $

功能: 在DOS状态下启动程序时, 获取命令行中的其它信息.

格式: COMMAND $

注释: 返回在DOS提示符下调用该程序时, 除该程序名之外的其它内容 (以字符串形式), 如输入数据等等.

注意: 在Turbo Basic 集成环境中运行含有COMMAND $ 函数的程序时, 要先用Options 选单中的 Parameter line 选项为 COMMAND $ 函数设置要获取的字符串.

COMMAND

功能: 说明要传递给被链接程序的变量

格式: COMMAND Variable [, variable]…

注释: Variable 是一个可传递到后面被链接程序的数组或标量变量. 数组通过在其名字后跟一对放有维数的园括号来表示. COMMAND 语句可出现在程序中任何位置, 也可以不止一句, 但给定的变量只能出现一次. 在被链接的程序中必须有相匹配的 COMMAND 语句, 两者包含的变量名可以不同, 但变量个数、次序和类型必须一致.

COS

功能: 返回自变量的余弦值

格式: COS (X)

注意: X为数值表达式, 按弧度计

CSNG

功能: 将数值转换为单精度数

格式: CSNG (X)

注意: CSNG将整型、长整型和双单精度值转换为单精度值. X的范围为: −32768～32767.

CSRLIN

功能:　返回光标的纵向位置（行号）

格式:　CSRLIN

注释:　返回值范围为1～25间的整数。

CVI, CVL, CVS, CVD

功能:　将从随机文件中读取的字符串型数据转换为数值型

格式:　CVI（X1$）
　　　CVL（X2$）
　　　CVS（X2$）
　　　CVD（X3$）

注释:　X1$，X2$，X3$分别为二字节、四字节和八字节的字符串。CVI、CVL、CVS和CVD分别将相应字节数目的字符串转换为整型、长整型、单精度型和双精度型数值.

注意:　这些函数只能用于随机文件，不要和VAL函数混淆。

CVMD, CVMS

功能:　将从Microsoft格式的随机文件中读取的字符串型数据转换为IEEE格式的浮点数。

格式:　CVMD(X1$)
　　　CVMS(X2$)

注释:　X1$，X2$分别为4字节和8字节字符串。CVMD，CVMS分别将相应的字符串转换为IEEE格式的单精度和双精度浮点数.

注意:　这两个函数只能用于Microsoft格式的随机文件，不要与VAL函数相混淆.

DATA

功能:　为READ语准备数据

格式:　DATA Constant[,contant]…

注释:　Constant是数值型或字型串型常数．数值型常数可以是浮点数或整数，也可以以十六进制或八进制形式描述．字符串常数不必用引号括住，除非它们包含分隔符(逗号或冒号）或有特殊意义的头空或尾空．字符串常数可与数值型常数任意混合使用．例如:

```
DATA Taco, .79, Chicken Supreme, 2.29,__
```

"Dr. Pepper, large", .89

一个程序可以包含多个DATA语句，而且不必连续排列。各DATA语句可以包含的数据一个数仅取决于一行中允许的字符数。

DATE $

功能: 设置或获取系统日期

格式:

DATE $ = X $

或:

X $ = DATE $

注释: 第一种形式根据X $中的内容设置系统日期。X $必须为"mm—dd—yyyy"形式的字符串（mm表示月，dd表示日，yyyy表示年）。而第二种形式获取机器内部的当前日期，形式同上。

DECR

功能: 设置或获取系统日期

格式: DECR Variable [, size]

注释: Variable是一个数值型变量。

Size为任选项，表示将从Variable中减少的数值；它可以是数值表达式。如果size项被省去，则默认1。

DEF FN / END DEF

功能: 定义函数

格式: 定义单行函数时:

```
DEF FNidentifier [ (argument list) ] = expression
```

定义多行函数时:

```
DEF FNidentifier [ (argument list) ]
        [LOCAL variable list]
        [STATIC variable list]
        [SHARED variable list]
          :   语句组
          [EXIT DEF]
          [FNidentifier = expression]
END DEF
```

注释: identifier 是一个与函数相关的唯一名称（函数名），它必须符合标号和变量的命令约定，一个字母后跟任意数目的字母和（或数字）.

argument list 变元表）为任选项，是一系列用逗号分隔的形式参数. 这些形式参数只用于定义函数，与程序中同名的其它变量没有关系.

DEFINT, DEFLNG, DEFSNG, DEFDBL, DEFSTR

功能: 定义变量的缺省类型.

格式: DEF type letter range [, letter range] …

注释: type(类型）表示五种Turbo Basic 变量类型中的一种（简写）: INT, LNG, SNG, DBL 和 STR. letter range(字母范围）是单个字母字符（A 到 Z, 大小写均可），或字母范围（两个字母用破折号分隔; 如, A—M).

注意: 类型说明语句不能放在块结构语句中，因此过程或函数定义中不能用这几条语句. 此外，类型说明语句只对其后的程序有效，因而它们通常放在整个程序的最前面. 多个类型说明发生冲突时，以最后一种为准.

DEF SGE

功能: 为BLOAD, BSAVE, CALL ABSOLUTE, PEEK 和POKE语句定义数据段.

格式: DEF SGE[=numeric expression]

注释: numeric expression(数字表达式）的值域为0～65535. 缺省此项时，DEF SEG 语句使段值返回到最初的默认值; 默认段中的头 256 个字节包含程序运行时供系统使用的一些信息. 该段中有一个有特殊意义的地址偏移量是&HF4E; 往该地址 POKE 进去值将改变图形方式下屏幕显示的颜色. 在默认数据段中另一个有用的地址是 0 和 1, 这个字中放有 Turbo Basic 的字符串段地址.

DELAY

功能: 暂停一段时间

格式: DELAY seconds

注释: seconds(秒数）是一个数值表达式，表示Turbo Basic 在执行下条语句前将暂停的秒数.

注意: DELAY 语句可接受浮点数，可分辨的最短时间大约为0.054秒.

DIM

功能: 定义数组

格式: DIM {STATIC | DYNAMIC} Var (Subscripts) [, var(Subscripts)]…

或:

DIM {STATIC | DYNAMIC}var(min:max [, min: max]…)
[,var(min:max [, min:max]…)]…

注释: var为数组说明符（数组名）; subscripts(下标) 是一系列由逗号分隔的、确定数组维数的一个或多个整数表达式. DIM 语句说明数组变量, 并定义其大小和维数。

DO, LOOP

功能: 在其顶部或底部带有条件测试的循环结构

格式: DO {WHILE | UNTIL} expression]
⋮ 语句组 [EXIT LOOP]
[LOOP | WEND] [{WHILE|UNTIL} expression]

注释: expression 是一个整型表达式, 其中非零表示条件成立 (TRUE); 零表示条件不成立 (FALSE).

DRAW

功能: 在图形屏幕上画图

格式: DRAW string expression

注释: DRAW 语句根据String expression(字符串表达式) 中的命令在图形屏幕上画图. 从效果上看, String expression 是一个解释 DRAW"语言"的程序. DRAW 执行命令字符串在屏幕上生成图像, 与 PLAY 语句执行命令字符串产生音乐非常相似.

string expression 中包含的移动命令如下（相对于前一个参考点而言）:

命令	说明
Un	上移
Dn	下移
Ln	左移
Rn	右移
En	向右上方作对角移动
Fn	向右下方作对角移动
Gn	向左下方作对角移动
Hn	向左上方作对角移动

上表中n表示移动的距离; 实际移动的象素数由n乘上比例因子（用S命令设置）来决定.

Mx, y 命令既可做绝对移动又可做相对移动: 如果X前面有加号 (+) 或减号

(—)，为相对移动；否则，为绝对移动．例如:

```
DRAW "D10 R5 M30, 50"
```

先画一个"L"形，再画一条到点（30，50）的直线。
下面两个命令前缀可以放在移动命令之前；

B　移动但不画图。
N　移动后返回原先的位置。

还可以使用其它命令:
An　设置一个角，n为0时表示0度，1表示90度，2表示180度，3表示270度。
TAn　转动一个角，n的值可在—360到+360之间．n为正，则逆时针旋转；n为负，则顺时针旋转。
Cn　设置颜色n．在中分辨率，n从前面COLOR语句中选定的套色中选择一种颜色，值可从0到3；在高分辨率，n为0或1．0表示黑色，1表示白色．对增强的图形板的情况，请查阅PALETTE语句中可以使用的颜色号。
Sn　设置比例因子．n的范围为1到255，如n为8，则比例因子为2，用此命令设置的比例因子，乘上U、D、L、R、E、F、G、H以及相对移动的M命令中给定的距离，就可以决定实际移动的距离．n的默认值是4，即比例因子为1.
Pcolor，boundary从现行坐标（x,y）开始，用color表示的颜色进行填充，遇到用boundary表示的颜色边界就停止．color和boundary都是数字，表示颜色号。
在DRAW语句中，可以直接使用字符串变量．如：DRAW "U20" 和 s$ ="U20"：DRAW s$ 的效果一样；也可以使用PC BASIC中的X命令:

```
10 LET s$ = "U20"
20 DRAW "X"+VARPTR$ (s$)
```

效果同上。
DRAW语句中如使用数字变量，必须在命令字母后加等号，并用VARPTR$(x)的格式，如:

```
A=20 : DRAW "U="+VARPTR$ (A)
```

效果同上。
举例:　下面的程序是在屏幕中间画一个房了，再把LPR（参考点）移到房子内部，然后用灰色（1号色）填充它；填充工作的边界是前面画房子时用过的白色（3号色），最后在屏幕顶部画五个房子。

```
SCREEN 1
HOUSE$ = "U20 G5 E20 F20 H6 D20 L30"
DRAW HOUSE$
DRAW "BE3"
DRAW "BE3"
DRAW "P1, 3"
FOR N%=1 TO 200 STEP 40
DRAW "BM="+VARPTR$ (N%) +", 40"
DRAW HOUSE$
NEXT N%
```

END

功能: 中止程序或定义一个结构块的末尾

格式: END [{DEF | IF | SELECT | SUB}]

注释: 无任选项时，END中止程序的运行。END语句可安放在程序中的任何地方，并且可有多于一个的 END. 执行 END 时，关闭所有打开的文件，并回到 DOS(或 Turbo Basic，根据程序被启用的地方而定)。

ENDMEM

功能: 返回物理存储器结束处的地址

格式: ENDMEM

注释: ENDMEM的返回值为长整型，表示内存中可用的最后字节所在地址. ENDMEM 函数与 ENDMEM 语句配合使用，可在存储器的高端为汇编语言子程序分配空间.

执行一个程序时，ENDMEM实际上返回的是: 减去符号表体积之后的物理存储器末尾地址. 在内存中进行编译时，符号表存储在内存地址的最高端.

ENVIRON

功能: 修改环境表中的信息

格式: ENVIRON x$

注释: X$ 即定义了要修改，增添或删除的参数名，又定义了新的参数内容. 二者必须用等号分隔. ENVIRON 语句与相应的 ENVIRON$ 函数通常用于设置环境参数. 执行 SHELL 语句时，这些环境参数就被传给这个程序 (参见 SHELL 语句和 DOS 手册中的 SET 命令). 例如:

```
ENVIRON "PATH=\TURBOBAS"
```

将把DOS的PATH参数置成"\TURBOBAS"。如果后面用SHELL 命令调用COPY 命令，将到\TURBOBAS 子目录中查找当前目录中找不到的文件。

如果要从环境表中删去一个参数，则只需在等号后加一个分号即可；如，

```
ENVIRON "PATH=;"
```

将从环境表中删去路径参数。

注意: 不能用该语句扩充环境空间。要想加入新项，必须删除已有的项，以便得到多余的空间。因此，如果未在DOS 状态下用SET 命令设置过任何环境参数，就不能使用该语句。

ENVIRON $

功能: 获取环境表中的参数

格式: ENVIRON $ (X $)

或:

ENVIRON $ (n)

注释: X $ 是参数字符串，它含有要获取的参数名。

n是1～255之间的整型表达式。

如果ENVIRON $ 的变元是字符串，返回的是环境表中该字符串后的内部，若找不到给出的字符串，或等号后元内容，则返回空字符串。当 ENVIRON $ 中的变元为数字时，返回的是从环境表起始处开始算起的第 n 个参数。若无第 n 个参数，则返回一个空串。

注意: 该函数要区分字母的大小写。

EOF

功能: 返回文件结束状态

格式: EOF(filenum)

注释: 文件指针指向文件结束处 (End Of File) 时，该函数返回逻辑值: 非0；否则，返回 0。filenum 为打开的文件号。

ERADR

功能: 返回最近发生错误处的地址

格式: ERADR

注释: 函数返回一个长整型数值。当错误捕捉到一个运行时错误时，函数ERADR 将给出程序计数器的值。程序可以把这个值报告给用户。用户可在 Debug 菜单中用 Run-time error 选项显示出程序中错误发生的位置。

ERASE

功能: 删除动态数组，初始化静态数组。

格式: ERASE array name [, array name]···

注释: array name 为要初始化或释放的数组名。如果数组为动态的，ERASE 语句则将其从内存中删除，释放它占用的内存；若数组为静态的，ERASE 语句将它置 0 或空字符串。

注意: 数组名后不跟括号。

ERDEV

功能: 返回设备错误的类型

格式: ERDEV

注释: 出现设备错误时，ERDEV 返回其最低有效字节中的出错信息。此字节表示INT 24 的错误代码。高位字节中含有设备属性字的第 15,14,13,3,2,1 和 0 位。返回值为整型。

ERDEV $

功能: 返回发生错误的设备名称。

格式: ERDEV $

注释: 返回值为字符串型，其中含有设备的名字：字符设备为8个字节，块设备为2个字节（参见 DOS 手册）。

举例:

```
        ON  ERROR  GOTO  ErrorHandler:
        PRINT  "Open the door to drive A and press any key "
        WHILE  NO  INSTAT:  WEND      'wait for a key
        FILES  "A:  *  *"
        END

ErrorHandler:
        'display an error message indicating problem and source
        PRINT USING "Critical error ## occurred with &";  ERDEV,  ERDEV $
        RESUME NEXT
```

程序运行时，如果关上A驱动器的门，将显示它的文件目录；否则，显示：

```
Critical error 2 occurred with A:
```

ERL

功能:　返回最近发生错误的行号

格式:　ERL

注释:　ERL 返回一个整型值，代表错误发生处的程序语句的行号. 如果包含错误的语句无行号，则返回最近执行的有行号语句的行号. 若计算机执行完全无行号的程序，则返回 0.

ERE

功能:　返回最近发生错误的错误号

格式:　ERR

注释:　ERR返回整型值，表示错误类型号.

ERROR

功能:　模拟运行时错误

格式:　ERROR n

注释:　ERROR 强制生成其类型由数字n代表的错误. n为0～255间的整数表达式.

EXIT

功能:　提前退出一个结构

格式:　EXIT {SELECT| DEF |FOR |IF| LOOP| SUB|WHILE}

注释:　EXIT 语句自结构化语句、过程或函数定义体中的某处（不是结束处）退出. 被提前 EXIT(退出）的结构类型必须按下表包括在 EXIT 语句中:

EXIT 选项	退出的结构
SELECT	SELECT 语句
DEF	函数定义
FOR	FOR / LOOP 循环
IF	IF 块
LOOP	DO / LOOP 循环
SUB	过程定义
WHILE	WHILE / WEND 循环

在要用GOTO语句的地方，用EXIT语句替代.

EXP, EXP2, EXP10

功能:　指数运算

格式:　EXP (X)

　　　　EXP2 (X)

　　　　EXP10 (X)

注释:　上述三个函数分别计算 e^x、2^x和10^x. x为数字表达式, e为自然数 (约为2.71828).

FIELD

功能:　定义随机文件缓冲区的字段变量

格式:　FIELD [#]filenum, width AS String—Var[, width AS String—Var]…

注释:　filenum 为文件被打开时使用的文件号;

　　　　width 为分配到字段变量的字节数;

　　　　String—var 为字段变量, 其类型为字符串型.

FILES

功能:　显示目录内容 (类似于DOS命令中的DIR命令)

格式:　FILES [filespec]

注释:　filespec 为符合DOS命令约定, 并可能包含驱动器说明符和路径信息的字符串表达式. 如果省略 filespec, 显示当前目录中的所有文件. filespec 可使用通配符: "*"和"?".

FIX

功能:　截去小数部分, 取整.

格式:　FIX (X)

注释:　X为数值表达式; 返回值为整型.

FOR / NEXT

功能:　定义一个自动增值 (或减值) 的循环

格式:
```
FOR Variable = x TO y [STEP Z]
  :     语句组
NEXT [variable [, variable]…]
```

注释:　variable 为用作循环计数器的数值型变量.

　　　　x, y和Z 为确定循环起始和终止条件的数值表达式. X为计数器的初始值; y 是

计数器的终值；Z 为任选项，表示增量（步长）.

FRE

功能：返回可供程序使用的剩余内存空间大小.

格式：FRE（{X$ | −1 | −2}）

注释：FRE（X$）返回字符串空间中剩余的字节数；FRE（−1）返回数组空间中剩余的字节数；FRE（−2）返回堆栈空间中剩余的字节数. X$ 一般置成" "（空串）.

GET(文件)

功能：从随机文件中读取一条记录

格式：GET [#]filenum [, recnum]

注释：filenum 为被打开文件的文件号；

recnum 是将读取的记录号，其数值范围为1～16777215(即2^{24} 1)．如果recnum被省略，那么顺序读取下一条记录（紧随最近被 GET 或 PUT 语句指定的记录）.

GET(图形)

功能：将图形屏幕的整体或部分存入数组

格式：GET(X1, Y1) — (X2, Y2), array

注释：(X1, Y1) 和 (X2, Y2) 是要存放的图形屏幕的矩形区域的左上角和右下角；array 是用于存放屏幕数据的数组，它是数值型的.

数组的大小必须足以放得下该区域中的数据，如果矩形的水平边长为X个象素，竖直边长为 Y 个象素，则该区域中数据所占用的字节数为：

4 + INT(X * n+7) / 8) * Y

其中n是每个象素占用的位数，在中分辩率时为2，在高分辩率时为1．例如在中分辩率时，要把宽为 35 个象素，高为 45 个象素的矩形区域存入数组，所需字节数最少为：4+INT((35 * 2+7) / 8) * 45，即 409 个字节．如果采用整型数组，则需要 205 个数组元素（每个整型元素占用两个字节)，可以使用下面的语句进行数组说明：

```
DIM buffer% (204)
```

在Turbo Basic 中按如下方式存储图形数据：

```
buffer% (0)      存放水平象素的个数
buffer% (1)      存放竖直象素的个数
  ……            存放图形数据
```

```
buffer% (204)
```

GET语句常与PUT语句配合使用，PUT语句可把数组中存放的图形画到屏幕上指定的位置．如果 X 的值是 4(在中分辨率时)或 8(高分辨率时)，可以使基于 GET 和 PUT 的动画更快一些．

举例: 下面先在屏幕上画一个园，再用GET语句将其存入一个整型数组buffer%，最后用 PUT 语句在屏幕中间将它复制 10 份．

```
DIM Buffer%(143) : SCREEN 1
CIRCLE (16,16),8,2
GET (0,0)-(31,31),Buffer%
FOR N% = 0 TO 9
    PUT (N% * 32, 100),Buffer%
NEXT N%
```

GET $

功能: 从二进制文件中读取数据

格式: GET $ [#]n, Count , X $

注释: n为文件号; count 为读取的字节数; X $ 为存放读出数据的字符串变量．读取的起点为当前文件指针所在的位置．

GOSUB

功能: 调用子程序

格式: GOSUB Label

注释: GOSUB 语句使Turbo Basic 跳到labe(标识的语句处，并将当前地址在入堆栈保存．执行 RETURN 语句时，使程序流程返回到紧随 GOSUB 之后的语句．

GOTO

功能: 使程序的流程转到标号处．

格式: GOTO Label

注释: GOTO 语句使程序流程无条件转向label(标号) 标记的语句处．

注意: 应避免使用GOTO 语句

HEX $

功能: 将十进制数转换为相应的十六进制格式字符串

格式: HEX $ (n)

注释:　n为—32768～65535的整型表达式．如果n为负值，则返回它的补码，返回值为字符串型．

IF(单行)

功能:　测试条件，并在条件满足时改变程序流程

格式:　IF 整型表达式[,]THEN 语句组[ELSE 语句组]

注释:　如果整型表达式的逻辑值为TRUE(即为非0时)，则执行THEN 之后至任选项 ELSE 之前的语句；如果整型表达式的逻辑值为 FALSE(结果为 0)，那么执行 ELSE 之后的语句．假如任选项 ELSE 子句被省略，继续执行程序的下一行．

IF（块)

功能:　创建一系列IF语句

格式:
```
IF 整型表达式 [,] THEN
    :     语句组
[ELSE IF 整型表达式 [，] THEN
    :     语句组]
[ELSE
    :     语句组]
END IF
```

注释:　IF块是解释型BASIC的扩展，允许在多行中进行多个条件测试．

INCR

功能:　增大变量

格式:　INCR numeric variable [，size]

注释:　Numeric variable(数值型变量）是要增大的变量．

Size为任选项，表示要加到数值变量上的数值；它可以是数值表达式．如size被省略，默认 1。

INKEY $

功能:　从键盘上读取字符串，但不显示．

格式:　INKEY $

注释:　返回由0,1或2个字符组成的字符串．这些字符串反映了键盘缓冲区的状态:

LEN $ （INKEY $）＝0(空串)，表示此缓冲区是空的．

LEN $ （INKEY $）＝1，表示返回最近所敲击的ASCII键符．

LEN $ （INKEY $）＝2，表示最近所敲击的键为非ASCII键（即扩展键，如 Home，PgDn 键)．此时，返回值中第一个

字符为 CHR$ (0)，第二个为 CHR$ (n)；
其中 n 为非 ASCII 键的扩展键码。

如果敲击了一个已定义的功能键（参见KEY语句），INKEY$将返回定义的字符串。

INP

功能: 从I/O端口读入一个字节

格式: INP(portno)

注释: portno 为I/O端口号 (0～65535)；返回一个0～255之间的整数。

INPUT

功能: 提示用户为变量赋值

格式: INPUT [;] [prompt string {; | ,}] variable list

注释: prompt string 为提示字符串，可有可无。
variable list 是由逗号分隔的一个或多个字符串或数字变量序列。
提示字符串后如果带有逗号，就不显示问号；如果提示字符串后跟分号，则用回车结束数据输入后，光标仍停留在同一行上。

INPUT#

功能: 从顺序文件中将数据读入变量

格式: INPUT #filenum, variable list

注释: filenum 是文件打开时给定的文件号；
variable list 是由逗号分隔的一个或多个字符串或数字变量序列。

注意: 文件可以是磁盘文件、串行口 (COMn:)，也可以是键盘 (KYBD:)。文件中的数据类型必须与 INPUT#语句中的变量类型相匹配。

INPUT$

功能: 从键盘或文件中读入指定数目的字符

格式: INPUT$ (m [, [#]n])

注释: m是要读入的字符个数；n为文件号。如果省略文件号，则从键盘读取。
如果从键盘读取字符，则读入的字符不显示，直接返回。与其它从文件或键盘输入的方法相比，INPUT$ 的主要优点在于它可接受包括控制字符在内的所有字符。

注意: 一些键，如功能键，没有ASCII码值；敲这些键时，代之以CHR$ (0)。

INSTAT

功能: 读取键盘状态

格式: INSTAT

注释: 当键盘缓冲器为空，即未敲任何键时，函数INSTAT返回0 (FALSE)；否则，敲了任一键时，返回—1 (TRUE)。可以在程序中包含如下形式的语句:

```
IF INSTAT THEN <操作>
```

以便仅当一次击键在键盘缓冲区中等待时，程序执行指定的操作。

INSTR

功能: 字符串查找

格式: INSTR([n,] target string, pattern string)

注释: 在target string (目标字符串) 中，从第n个字符开始查找patternstring(样板字符串)。若未找到，返回值为 0; 否则，找到时，返回样板字符串在目标串中的起始位置。如果样板串是个空串 (长度为 0)，则 INSTR 返回 1(或 n，指定了 n 时)。n 的缺省值为 1。

INT

功能: 将数值取整

格式: INT (n)

注释: INT 返回小于或等于n的最大整数。n为数值表达式。

IOCTL

功能: 与设备驱动器通讯

格式: IOCTL [#] filenum, string expression

注释: filenum 为期望设备驱动器的文件号。

string expression(字符串表达式) 为要发送给设备驱动器的信息。

KEY

功能: 设定、显示功能键的内容，以及定义击键陷井值。

格式: KEY {ON 1 OFF 1 LIST}

KEY n, string expression

KEY n, CHR $ (shift status) +CHR $ (scancode)

注释: KEY ON 和KEY OFF 开／关屏幕底部的功能键显示。注意，关闭功能键的显示并不影响它的定义。KEY LIST 将当前功能键的定义传送到屏幕显示。显示功

能键时，屏幕的第 25 行从不翻卷，并且将光标（用 LOCATE）定位在这一行的试图将导致第 5 号运行时错误：Illegal Function Call.

KEY n，string expression 把第n号功能键设置为String expression (字符串表达式）指定的字符串．字符串可长达 15 个字符，但最多显示 6 个．要取消某功能键的设置，可将它置为空字符串．字符串中的回车符显示为一个向左的小箭头．第三种格式将一个键或键的组合与一个数字n（可从15到20）联系起来，以便后续的 ON KEY 语句和 KEY(n) ON 语句对它进行捕获．shiftstatus 是 0 到 255 之间的整数，它控制着对几个特殊键进行捕获：Ctrl，Caps Lock，Num Lock，Alt 和两个 Shift 键，scancode 是 1 到 83 之间的一个数，它用于根据键盘扫描码来定义捕获用的键．

注意：59到68以及72，75，77 和80 等功能键和光标控制键已经被定义了，所以再对它们定义也是无效的，可以用下列值定义 shiftstatus:

Modifier key	二进制值	十六进制值
右边的 Shift	0000 0001	01
左边的 Shift	0000 0010	02
Ctrl	0000 0100	04
Alt	0000 1000	08
Nun Lock	0010 0000	20
Caps Lock	0100 0000	40

举例：假设要对"Shift—Escape"进行捕获，即希望不论什么时候，只要按住Shift 键时又击了 Esc 键，就转去执行某段子程序．首先对两个 Shift 键进行屏蔽 01H+02H＝03H，然后查出 Esc 的扫描码是 1，所以 KEY 语句应为：KEY 15，CHR＄（3）+CHR＄（1）

再定义捕获Shift—Esc用的陷井；（键值只能为15，因为前14个值已预定义为功能键和光标控制键）．

```
ON KEY (15) GOSUB ShiftEsape    'ShiftEsape 为子程序的标号
```

再打开对键15 的追踪捕获功能：

```
KEY(15) ON      ' 注意，使用时Caps Lock 键要置为小写字母状态．
```

KEY (n)

功能：开／关某特定键的击键陷井

格式：KEY (n) {ON OFF STOP}

注释：n 是 1～20间的整数表达式，表示设陷的键号．n值与键的对应关系如下：

n	键
1～10	十个功能键： F1～F10
11	光标上移键
12	光标左移键
13	光标右移键
14	光标下移键
15～25	由 KEY 语句定义的键
30	F11 功能键
31	F12 功能键

KEY（n） ON 启动对键n的捕获。这就意味着在各条语句间都要进行检查，看是否键 n 被按。如果键 n 被按，则程序转向执行 ON KEY 语句中为该键指定的子程序。

KILL

功能： 删除文件（类似于DOS中的DEL命令）

格式： KILL filespec

注释： filespec 是一个表示待删除文件的字符串表达式。其中可包括路径名和／或通配符。KILL 语句类似于 DOS 的 DELETE(ERASE) 命令。

LBOUND

功能： 返回数组某维的最小下标。

格式： LBOUND(array(n))

注释： array 为数组名；n为数组的维数号。

LCASE $

功能： 将字符串中的字母全部变为小写

格式： LCASE $ （X $ ）

注释： X $ 为待转换的字符串。返回值为字符串型。

LEFT $

功能： 获取字符串最左边的n个字符

格式： LEFT $ （X $ ，n）

注释： 当n大于字符串X $ 的长度时，返回X $ ；n应为大于或等于0 的整数，否则将导致运行时错误。

LEN $

功能: 求字符串的长度

格式: LEN $ (X $)

注释: X $ 为待求的字符串; 返回值为大于或等于0的正整数.

LET

功能: 为变量赋值

格式: [LET] variable = expression

注释: variable 是一个字符串型或数值型变量;

expression 是一个相应类型的表达式（即, 对于字符串型变量, 为字符串; 而对于数值变量, 则为数值).

LINE

功能: 画直线、方框或方块

格式: LINE [(X1, Y1)]— (X2, Y2), [,[color] [, B[F]] [, pattern]]

注释: (X1, Y1) 和 (X2, Y2) 为图形屏幕上两点的坐标, 既可以绝对值形式, 也可以相对值形式指定.

color(颜色) 为描述线条或盒子将配置颜色的整数表达式.

pattern(图案) 是一个0～32767之间的整数, 决定线条或盒子绘制时所用的花样.

LINE INPUT

功能: 从键盘上把一行数据赋给一个字符串变量, 忽略定界符.

格式: LINE INPUT [;] [prompt string ;] string variable

注释: prompt string(提示字符串) 是一个任选项, 表示在等待用户响应之前发送到屏幕上的字符串常数 (提示信息).

String variable(字符串变量) 是键盘上键入数据的存放变量.

不得不输入包括定界符 (即, 逗号) 的字符串信息时, 应用LINE INPUT 语句代替 INPUT 语句, 否则将引起 INPUT 语句的混乱. 例如:

```
INPUT "Enter Patient address: "; a $
```

如果输入的地址中含有逗号, 则:

```
Enter Patient address:  101 Main Street , Apt 2
? Redo from start
```

INPUT 后的分号可使执行该语句后，显示不换行.

LINE INPUT

功能: 从顺序文件中将一行数据读入一个字符串变量; 忽略定界符.

格式: LINE INPUT #filenum, String variable

注释: filenum 为要读取数据的文件号.

String variable 是数据将装入的字符串变量.

LOC

功能: 返回文件指针的位置（当前位置）

格式: LOC (n)

注释: n是文件号. 返回值随文件类型而不同:

对于通讯文件，返回值表示输入缓冲区中字符的个数.

对于随机文件，返回值表示刚读入或刚写出的记录号.

对于二进制文件，返回值表示文件中的位置（以字节为单位）

对于顺序文件，返回值表示自打开文件以来读/写的块数（每块128个字节）. 刚打开文件时，块数为 1.

LOCAL

功能: 在过程或函数内说明局部变量

格式: LOCAL variable list

注释: variable list(变量表)为一系列用逗号分隔的变量名和/或数组名.

在一个过程的定义或自定义函数段的开始，可以利用LOCAL 语句将变量说明为: 对于该过程或函数来说，是局部的. 局部变量与过程或函数以外的同名变量毫无关系，所以在给它们命令时无需考虑"外部"的变量. 当程序引导到该过程或函数时，计算机仅为局部变量分配内存; 而当程序退出时，再释放这些内存. 因而，在过程或函数的相继引用之间，局部变量不保留它们的值.

也可以用LOCAL 将数组说明为局部的，这首先要在LOCAL 语句中给出它的名字，并在后面跟着空括号，然后在后面的 DIM 语句中定维.

LOCATE

功能: 给光标定位和/或定义光标的形状

格式: LOCATE [row] [,[column] [,[cursor] [, start][, stop]]]

注释: row是一个整数表达式，确定光标将安放的屏幕行（1～25).

column 是一个整数表达式，指定光标将安放的屏幕列（1～80).

cursor(光标）是一个数值（0或1)，控制光标是否可见（0表示不可见，1表示可

见）.

start 和stop 都是整数表达式，用于控制光标的大小，即光标由几个扫描行所组成．对于彩色图形板，光标最上端为第 0 号扫描行，最下端为第 7 号扫描行；对于单色显示板，光标最下端为第 13 号扫描行．

注意：除非功能键显示被消隐，否则，屏幕上的第25行不能用（参见KEY OFF 语句）．

LOF

功能：返回磁盘文件的长度或通讯文件的通讯缓冲区中剩余可用空间的大小．

格式：LOF(n)

注释：n 为文件号．

LOG，LOG2，LOG10

功能：进行对数运算

格式：LOG（X）

LOG2（X）

LOG10（X）

注释：X为数值表达式，其结果必须为正数．LOG，LOG2和LOG10分别计算以自然数(e)为底，以 2 为底，和以 10 为底的对数．

LPOS

功能：获取打印缓冲区的当前位置．

格式：LPOS(printer)

注释：print 是0～3之间的一个整数，代表各打印机：

printer	打印机
0, 1	LPT1:
2	LPT2;
3	LPT3:

LPOS 函数的返回值表示自上一次回车以来，又向该打印机发送了多少字符．

LPRINT，LPRINT USING

功能：向打印机（LPT1:）发送数据

格式：LPRINT [expression [;]]

LPRINT USING format string； expression list

注释:　expression 是一系列用逗号或分号分隔的数值或／和字符串表达式.
format string 包含格式信息，(参见PRINT 和PRINT USING 语句).

LSET

功能:　将字符串数据放入随机文件缓冲区.

格式:　LSET field variable = string expression

注释:　LSET 和它的姐妹语句RSET 将字符串信息放入作为随机文件缓冲区附属量定义的字段变量中.

如果String expression（字符串表达式）的长度比在FIELD 语句中指定的fieldvariable(字段变量）体积小，LSET 则通过插入空格的方法将字段变量左对齐. 这就是说，在 LSET 语句赋值之后，空格将添加到 string expression(字符串表达式）的后面. 字段变量的长度将一直等于 FIELD 语句中定义的宽度.

RSET 语句用插入空格的方法实现右对齐（空格加在String expression 的前面).

LSET 语句还可用于对屏幕和打印机进行格式化输出（参见RSET语句).

MEMSET

功能:　规定内存的上限

格式:　MEMSET address

注释:　address 是一个长整型表达式，确定Turbo Basic 内存上限的绝对地址. address 必须小于系统安装的内存总量. 实际上，address 通常是用 ENDMEM 函数的返回值减去一个常数的方法得到. MEMSET 语句用于为汇编语言子程序单独留出一块内存空间.

举例:

```
PRINT FRE(-1)              '显示可用的数组空间
MEMSET ENDMEM—&H80    '在内存的顶端保留 128 字节的空间
PRINT FRE(-1)              '显示可用的内存空间
```

MID $

功能:　替换字符串／自字符串中提取子字符串

格式:　MID $ (String Variable, n[,m]) = replacement string

MID $ (string variable, n[,m])

注释:　第一种形式用replacement string(替换字符串）替换String variable(字符串变量）中自第 n 个字符开始的 m 个字符;

第二种形式自string variable 中提取从第n个字符开始的m个字符，作为一个函数使用.

n和m分别是1～32767和0～32767之间的数字变量或数字表达式. 如果省略了m

，那么操作进行到字符串变量 string variable 的末尾。

MKDIR

功能:　创建一个子目录（类似于DOS的MD命令）

格式:　MKDIR Path

注释:　path 是一个描述要创建子目录的字符串表达式（子目录名）。

MKI $, MKL $, MKD $

功能:　将数据转换为要输入到随机文件中的字符串。

格式:　MKI $ (a)

MKL $ (b)

MKS $ (c)

MKD $ (d)

注释:　MKI $，MKL $，MKS $ 和MKD $ 分别将相应的数值转换为2字节，4字节，4 字节和 8 字节的字符串；a、b、c、d 分别表示整型、长整型、单精度浮点型和双精度浮点型数值表达式。

MKM $, MKMD $

功能:　将数据转换成Microsoft 格式的字符串（用于随机文件的输出）

格式:　MKI $ (a)

MKD $ (b)

注释:　a,b分别为单精度和双精度型数值表达式。这两个函数用于Microsoft 格式的随机文件。

MTIMER

功能:　读取／重新设置微计时器

格式:　MTIMER

注释:　MTIMER 最初是用来测量过程很短的操作所耗费的时间。其精度较高(在54毫秒之内，约为 2 微秒)；不过，测量 54 毫秒之内的时间时，其精度急剧下降。

MTIMER 既可作为一条语句，也可作为函数使用。MTIMER 用作语句时，将微时钟计时器重新置零；MTIMER 用作函数时，返回自最近一个 MTIMER 语句被执行以来所经过的时间（按微秒计）。

实际使用时，应先招待MTIMER 语句。将微计时器置为零，再调用MTIMER 函数以获得经过的时间。

调用一次MTIMER 函数之后，如果没有用MTIMER 语句重新进行设置，再调用此函数将得到零。

MTIMER 和SOUND、PLAY语句都要使用8255片子上的2 号通道，并且是相互排斥的，所以在 MTIMER 语句和 MTIMER 函数之间执行 SOUND 或 PLAY 语句都会使微计时器重新置为零.

NAME

功能：为文件改名（类似于DOS的REN命令）

格式：NAME filespec1 AS filespec2

注释：filespec1 和filespec2 都是符合DOS 路径和文件名约定的字符串表达式. NAME 将 filespec1 表示的数据（文件名）命名为 filespec2。如果 filespec2 包含路径名，那么将数据（文件）由一个目录移至另一个目录是可能的，只要不试图自一个磁盘改名到另一个磁盘就可以.

OCT $

功能：返回数值的八进制形式字符串.

格式：OCT $ (n)

注释：n 为—65535～65535之间的整型表达式. 当n为负数时，返回n 的补码对应的字符串.

ON COM (n)

功能：设置通讯陷井

格式：ON COM (n) GOSUB label

注释：n 为1 或2，表示要读取的通讯适配器（串行口）的号码.

label(标号）标识陷井捕捉子程序. 如果label 为特殊行号：0，那么关闭陷井捕捉. 计算机执行 COM (n) ON 之后，信息进入指定端口便引起一个转子过程.

ON ERROR

功能：设置错误陷井

格式：ON ERROR GOTO label

注释：label 标识错误捕捉程序的第一行.

ON / GOSUB

功能：根据数值表达式的值调用几个可能的子程序中的一个.

格式：ON n GOSUB label [, label]…

注释：n为0～255之间的数值表达式；label 标识一条要转移到的语句。本书避免使用转

移指令，包括 ON／GOTO....语句体，执行时，若 n 为 0 或大于标号的个数，则执行下一条语句；否则转移到第 n 个标量标识子程序。

ON／GOTO

功能: 根据数值表达式的值，将程序流程转到几个可能的程序段中的一个。

格式: ON n GOTO Label[, Label] ...

注释: n 是0～255之间的数值表达式；Label标识程序中一条要转移到的语句。执行时，若 n 为 0 或大于 label 的个数，则执行下一条语句；否则，转向第 n 个 Label 标识的程序段。

ON KEY (n)

功能: 如果一个特定的键被按，ON KEY(n) 语句便执行陷井捕获子程序。

格式: ON KEY (n) GOSUB label

注释: label标识陷井捕获子程序的第一条语句。

n为1～20间的整型表达式，表示将捕获的键，n与各键对应的关系如下:

n	键
1～10	F1～F10
11	↑
12	←
13	→
14	↓
15～25	KEY 语句定义的键
30	F11
31	F12

ON PEN

功能: 如果发生光笔活动，ON PEN 语句便执行陷井捕获子程序。

格式: ON PEN GOSUB Label

注释: label标识光笔处理子程序中的第一条语句。

ON PLAY

功能: 如果背景音乐缓冲区包含的音符数目少于指定数目，ON PLAY 语句便执行陷井捕获子程序。

格式: ON PLAY(notecount) GOSUB label

注释:　notecount是一个整数表达式，指定设陷的音符数目.

　　　label 标识音乐演奏子程序的第一条语句.

举例:

```
$SOUND 80                              '将音乐缓冲区设置为 80 个字节，10个音符
ON PLAY(2)  GOSUB FillplayBuffer       '设置音乐陷井
PLAY ON                                '使音乐陷井生效
PLAY "MB AAAA"                         '演奏四个相同的音符，使音乐缓冲区有音符
WHILE  INKEY$ = ""  WEND               '击任意一键后才停止运行
END

FillPlayBuffer:                        '音乐子程序
PlayStr$ = "MB CDEFGAB"                '用此字符串表示一段音乐.
PRINT  PLAY (1)                        '显示音乐缓冲区中剩余的音符数.
PRINT  PlayStr$                        '显示下一句要演奏的乐曲.
PLAY PlayStr$                          '演奏此字符串表示的乐曲
PRINT  "Press any key to END"          '提示用户怎样停止程序
RETURN
```

由于每次从子程序返回后仍处在WHILE的循环内，所以总不停地进行捕获，只有当用户击了某一个键后，主程序才会结束.

ON STRIG

功能:　设置游戏棒陷井

格式:　ON STRIG (n)　GOSUB label

注释:　label 标识"按游戏棒"处理子程序的第一条语句；n是整型表达式，表示设陷的按钮。n 与按钮的关系如下:

n	按钮
0	按钮 1，游戏棒 A
2	按钮 1，游戏棒 B
4	按钮 2，游戏棒 A
6	按钮 2，游戏棒 B

ON TIMER

功能:　每隔n秒便执行陷井捕获子程序一次.

格式:　ON TIMER(n)　label

注释:　label 标识"时间流逝"处理子程序的第一条语句；

n为整型表达式，表示等待的秒数，其数为: 1～86400秒（24小时）.

OPEN

功能: 为读／写操作准备一个文件或一台设备

格式: OPEN filespec [FOR mode] AS [#]Filenum [LEN =record size]

或:

OPEN mode string, [#]filenum , filespec [record size]

注释: mode(工作模式)是下列方式关键字之一:

OUTPUT	指定一个要存写的顺序文件
INPUT	指定一个要读取的顺序文件.
APPEND	指定一个要附加的顺序文件.
RANDOM	指定一个要读或写的随机文件.
BINARY	指定一个要读或写的二进制文件.

mode string 是一个首字符如下所述的字符串表达式（通常仅含有首字符):

O	指定顺序输出模式
I	指定顺序输入模式.
A	指定在文件尾部顺序写的模式
R	指定随机输入／输出模式
B	指定二进制输入／输出模式

filenum(文件号）为任意整型值. 在CONFIG.SYS文件中用DOS的FILES语句加大或减小 filenum 的最大可用值.

filepec(文件说明）是一个字符串表达式，指定将打开文件的名字，其中也可指定驱动器和／或路径.

record size(记录长度）是随机文件中一条记录的长度，以字节为单位，其数值范围为: 1～32767. 省略时，默认 128 字节. 该任选项仅对随机文件有效.

OPEN COM

功能: 打开并配置一个通讯端口.

格式: OPEN "COMn: [baud] [, parity][, data][, step][, options]" AS [#]m [LEN = num]

注释: n的值是1或2，用于指定要找开的串行通讯口.

baud是一个整常数，用于指定通讯的速率. 有效的速率为: 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 和 9600. 默认为 300.

parity 是一个字符，用于指定发送和接收的奇偶性，可使用的字符如下:

S	奇偶位总是作为空格（0）发送和接收。
O	奇数奇偶校验。
M	奇偶位总是作为标志（1）发送和接收。
E	偶数奇偶校验。
N	不进行奇偶校验（接收时不理睬奇偶校验，发送时省略奇偶校验）。省略此项时，默认 E。

data 是5到8之间的一个整常数，用于指定数据位的个数，默认为7。
stop 是1或2，用于指定停止位的个数，默认为1，波特率75和110为2）。
m 是一整数，表示文件号；机器将通过此文件号访问该通讯口。
num 是一个整数，用于指定GET语句可从该通讯口读取的最多字节数，或用PUT 语句可向该通讯口发送的最多字节数，默认值为 128，这个值不能大于用 $ COM 伪语句为该串行口设置的值。

```
options = [, RS] [, CS [msec]] [, DS[msec]] [, CD [msec]] [, LF] [, PE]
```

options的一系列选择项用于控制线路状态。各项的含义如下：

RS	禁止（RTS）请求发送
CS	控制 CTS
DS	控制 DSR
CD	控制 CD
LF	在回车符后自动加上换行符。
PE	允许检测奇偶错误。

CS, DS 和CD 后的msec 的值在0到65535之间，单位是毫秒，用于指定等待有关信号的时间；如果在指定时间内检测不到有关信号，将产生设备超时错误（Device Timeout error）。如果省略 msec，或 msec 的值为 0，则根本不检测有关的信号；如省略 CS，DS 和 CD，则默认为 CS1000，DS1000，CD0。

举例：

```
OPEN "COM1:" AS #1
```

打开串行口COM1：作为1号文件，波特率为300，偶数奇偶校验，7个数据位，1个停止位。
又如：

```
OPEN "COM1: 9600, N, 8,, CS, DS, CD," AS #1
```

以9600的波特率打开COM1：，无奇偶校验，有8个数据位，不检查CTS，DSR和 CD.

下面的程序用于从某数字化仪（K—8410）接收数据（一对坐标），并将该点画到屏幕上.

从该数字化仪的使用说明书知道，它用串行ASCII码格式输出数据，每采一个点就输出由 15 个 ASCII 码组成的一个字符串：

CB ± XXXXX ± YYYYYCR LF

其中第一个字符是标志位，最后两个是回车符和换行符，中间是两个带符号的五位数，分别表示 X 和 Y 的坐标值．用 INPUT#语句可将这一字符串中的前 13 个字符输入给一个变量 A＄，再从中分离出标志位，X 和 Y 的值做进一步处理（此数字化仪有四个按键，按不同的键则标志位的值不同）.

先确定OPEN语句的参数．根据该数字化仪的使用说明书，它的波特率可为300，串行输出每个字节要用 7 个数据位和 2 个停止位．因此 OPEN 语句应为：

```
OPEN "COM1: 300, N, 7,2,RS, CS, DS, CD" AS #1
```

其中N表示不接收奇偶校检

程序如下：

```
REM    ' 让数字化仪工作于点方式、公制（比例因子为1）、输出为ASCII码
         串型
REM    ' 数字化仪的面板开关设置如下（1表示ON）：SW1: 11100
                SW2: 11111
SCREEN 2 : CLS
WINDOW(0.2800) — (4100.0)          ' 此数字化仪的有效幅面是410×
                                     280 毫米
INPUT "Are you ready ? [Y/N]", C$    ' 让用户做好准备
30 IF C$ < > "Y"AND C$ < > "Y" THEN GOTO 30
CLS: C$ = "2"                       ' 击红键采点，击其它颜色键退出
OPEN "COM1: 300, N, 7, 2, RS, CS, DS,CD" AS #1
WHILE C$ = "2"                      ' 击其它颜色键则 C$不为 2
   80 INPUT #1, A$
   IF LEN (A$) < > 13 THEN GOTO 80  ' 防止错误的输入码
   C$ = LEFT$ (A$, 1)               '取出控制字符
   LOCATE 2, 1
   PRINT "A$ = "; A$                ' 显示接收到的字符串
   X$ = MID$ (A$, 2, 6): X = VAL(X$)  ' 从 A$ 中取 X
   Y$ = MID$ (A$, 8, 6): Y = VAL(Y$)  ' 从 A$ 中取 Y
```

```
    PRINT USING "X=######"; X;
    PRINT USING "Y=######", Y
    PSET(X,Y) ,1                    ' 在屏幕上绘出此点
WEND
PRINT "OK"                          '现在可以做其它工作
```

注意: options 前面各项的位置是固定的; 省略某项时, 也要用逗号指明它后面项的位置. 而 options 内各项的位置是可变换的.

OPTION BASE

功能: 设定数组下标的下界

格式: OPTION BASE n

注释: n为0～32767之间的整数表达式.

OUT

功能: 将一字节输出到I/O端口

格式: OUT portno, integer expression

注释: portno为I/O端口号, 其数值范围为: 0～65535.

integer expression为要输出到I/O端口得值; 值域为: 0～255间的整数.

举例: 下面的程序通过读取状态寄存器的值, 并反复开/关控制扬声器的某些位使扬声器发声.

```
StatusReg% = INP[&H61)          ' 从 61 号端口读一个字节给 StatusReg%
StatusReg% = StatusReg% AND &H00FC   ' 将其最后两位置 0
FOR N = 1 TO 1000                ' 控制发声时间
    StatusReg% = StatusReg% XOR 2    ' 反复开/关最后两位
    OUT &H61, StatusReg%             ' 向扬声器发送控制信号
    DELAY 0.001
NEXT N
```

PAINT

功能: 在图形屏幕上用选定的颜色填充封闭区域

格式: PAINT (x, y) [,color][,boundary][,background]

注释: (x, y)是被填充区域内得一点.

color可以是一个数字或一个字符串. 如果是数字, 它就表示填充所使用的颜色; 如果是字符串, 就用字符串代表的图案进行填充. 若省略 color, 就用

前景颜色填充。

boundary是填充区域的边界颜色号，遇到此颜色就停止填充。

对于SCREEN 7, 8, 9, 10这些增强型的屏幕方式，使用下述方法设计图案：把字符串看作是由 8 位一个单元、8 位一个单元组成得一组数据来存储和处理，把这种位的集合称为“位面(bit plane)”。每种屏幕方式需要用 4 个位面定义一个图案，但方式 10 需要用 2 个位面。

举例： 下面程序在中分辨率图形方式下画两个园，并分别填以不同的图案：

```
SCREEN 1 : CLS
TIL1 $ = CHR $ (&H40)+CHR $ (&H40)+CHR $ (&H10)+CHR $ (&H10)_
          +CHR $ (4)+ CHR $ (4)+CHR $ (1)+CHR $ (1)
                                          ' 定义斜线图案
CIRCLE (100, 100), 50
PAINT (100, 100), TIL1 $
TIL2 $ = CHR $ (&HAA)+CHR $ (&HAA)+CHR $ (&H80)+CHR $ (&H80)_
        +CHR $ (&H80)+ CHR $ (&H80)+CHR $ (&H80)+CHR $ (&H80)
                                          ' 定义方格图案
CIRCLE (225, 100), 50
PAINT (225, 100), TIL2 $
END
```

PALETTE, PALETTE USING

功能： 改变调色板中的一种颜色或多种颜色(仅适用于EGA)

格式： PALETTE [attribute, color]

或：

PALETTE USING a%(index)

注释： attribute表示该调色板中的一种颜色。color表示要用来替换attribute的另一种颜色。实际的颜色与设置的屏幕方式以及硬件设备有关(详见 SCREEN 语句)。

PALETTE语句可用另一种颜色来改变屏幕上当前显示的字符和图形的颜色，所作出的改变也影响后续的图形和字符。只想改变字符和图形颜色时，就不必重写、重画了。

用PALETTE语句还可把大于15的颜色号映射到调色板中；可使显示的文字和背景色一样，成为看不见的；然后，改变文本的颜色，使之变成可见的。无参数的 PALETTE 语句把调色板置成默认的方式。

PALETTE USING语句可以修改调色板上的所有项目。其中a%表示一个整型数组名，index 表示下标。

设置调色板时，用index指明数组的第一个元素(多数情况下为0)。要保证数组能放下调色板的 16 个项目。调色板中的 attribute 和数组中的 color是配对的。用

-1 作为一个数组项的颜色号可以使与它对应的 attribute 保持不变。使用其他的负数都是不合法的。下表列出了对应于各种屏幕方式和显示器的 attribute 和 color 的取值范围:

模式	显示器	Attrbute 范围	Color 范围
0	Mono / EGA	0~15	0~2
	Color / EGA	0~31	0~15
1	Color / EGA	0~3	0~15
2	Color / EGA	0~1	0~15
7	Color / EGA	0~15	0~15
8	Color / EGA	0~15	0~15
9	EGA(64K)	0~3	0~15
	EGA(>64K)	0~15	0~63
10	Mono / EGA	0~3	0~8

Mono = Monochrome(单色显示器)

EGA = IBM Enhanced Graphics Adapter(IBM 增强型图形适配器)

举例:

```
' The following shows how to use PALETTE to change the color of graphics
' (and text)  without having to redraw or redisplay the graphics
SCREEN 8              ' set EGA 640 * 200 16 color mode
PALETTE               ' set default EGA palette colors
LINE( 10, 10) — (630, 190), 1, BF     ' draw a box , fill it with blue
DO
    FOR  N%=2  TO  14
        PALETTE 1, N%       ' change the box color from blue
        DELAY 2             ' delay , so you can see the change
    NEXT  N%
LOOP  UNTIL  INSTAT         ' do it until a kev is pressed
END
```

PEEK

功能: 返回指定存储单元的值 (一个字节的值)

格式: PEEK(offset)

注释: offset 表示在现行段中的偏移地址, 其段地址是由最近的DEF SEG 语句指定的, 得到的函数值是 0—255 之间的一个整数。

举例:

```
DIM Array% (5)
DEF SEG = VARSEG (Array% (0))
FOR N%=0 TO 11 : POKE N%, &HFF: NEXT N%     ' 对数组初始化
FOR N%=0 TO 11       ' 显示数组的值.
   PRINT HEX$ (PEEK (N%));
NEXT N%
```

PEN(函数)

功能: 读取光笔状态

格式: PEN(Option)

注释: Option 选项是一个数字表达式, 其值域为0—9; 返回值含义如下:

0 光笔片上次查询以来是否落下? 已落笔函数值为—1, 否则为0.
1 得到光笔最近移动的X坐标.
2 得到光笔最近移动的Y坐标.
3 得到光笔当前的开关值: 落下为—1, 抬起为0.
4 得到最后知道的X坐标 (0—319或0—639).
5 得到最后知道的Y坐标 (0—199) .
6 得到光行最近动作的 (字符) 行位置 (1—24).
7 得到光笔最近动作的 (字符) 列位置 (1—40或1—80).
8 得到最后知道的 (字符) 行位置.
9 得到最后知道的 (字符) 列位置.

举例:

```
ON PEN GOSUB PenHandier
PEN ON
PRINT "Press any key to stop "
WHILE INKEY$ = ""WEND
END

PenHandler:
   FOR N%=1 TO 9  :  PRINT PEN(N%)   : NEXT N%
   RETURN
```

PEN(语句)

功能:　控制光笔事件的检查

格式:　PEN { ON | OFF | STOP }

注释:　PEN 语句控制光笔陷井是否生效．在执行任何PEN 功能之前，必须先用PEN ON 打开光笔陷井．执行 PEN ON 语句也可通过在 ON PEN 语句中指定的子程序俘获光标文件．

PLAY(函数)

功能:　得到背景音乐缓冲区中音符的个数

格式:　PLAY(X)

注释:　X是个虚变元，即X 可为任意数而对函数值没有影响．PLAY 函数得到后台音乐缓冲区中等待演奏的音符的个数．如果没有音符，或音乐是由前台演奏，PLAY 函数将得到 0．

举例:

```
        REM   PLAY do-re-mi-fa-sol and count notes remaining
        PLAY "MB COEFG "
Again:
        n% = PLAY  (1)
        PRINT n%; "notes left in buffer
        IF n% < > 0 THEN GOTO Again
        PRINT " Buffer Empty"
        END
```

PLAY(语句之一)

功能:　控制音乐陷井的状态

格式:　PLAY {ON|OFF|STOP}

注释:　这种形式的PLAY 语句控制音乐事件的检查与否．在执行任何音乐陷井处理语句之前，应先用 PLAY ON 打开音乐陷井．

PLAY(语句之二)

功能:　产生音乐

格式:　PLAY string expression

注释:　string expression 为包含音乐演奏命令的字符串表达式。

PLAY 语句根据字符串表达式的值演奏音乐，该字符串是用一种小型的音乐语

言写成的。使用这种音乐语言，就可以用字符串表示一段音乐，用 PLAY 语句来演奏它。这和 DRAW 语句使用一种小型的绘图语言进行绘图是一样的。下面介绍这种音乐语言中音符的表示方法和各种命令。

与钢琴上的键相对应，这种音乐语言中共有84个音符，即从低于中C的第三个八音度开始，到高于中 C 的第四个八音度的前一个音符为止。

有两种表示音符的方法：用1到84之间的一个数字或用A到G之间的一个字母。

用数字表示音符时只需在数字前加上N命令，例如：PLAY “N44” 演奏高于中 C 的 G。

用字母表示音符时，在字母前应该用O命令说明该字母音符所在的八音度，例如：

PLAY "03 G"

将演奏第三个八音度中的G（与“N44 为同一个音）。在字母音符的后面，可以带升音或降音符。升音用“# ”或“+”表示。降音用“– 表示。但是这里的升音或降音必须和钢琴上的黑键相对应，例如“F–”就是不合法的。

下面介绍各条命令，这些命令用一个或两个字母表示。在PLAY 语句的字符串中，所有的空格都忽略不计，即各命令间有无空格是一样的。

Nn 表示音符n，n是个数字表达式，它的值可在0到84之间，1表示钢琴上最低的 C；84 表示钢琴上最高的 B；0 表示休止；中 C 是 37。

On 表示第n个八音度。n可在0 到6之间，每个八音度从C 开始，到B结束。中 C 是第三个八音度的头一个音。省略此命令，则默认为是第四个八音度。

>n 从现在使用的八音度上提高一个音度，再演奏音符n。

<n 从现在使用的八音度上降低一个音度，再演奏音符n。

Ln 为后面的音符规定时间，n可在1到64之间。

L1 全音符
L2 二分音符
L4 四分音符
……
L64 六十四分音符 可以使用1到64之间其它数，如L5等等。

Pn 休止n次，以前面用L 命令指定的时间为一次。

• 音符后面带一个圆点将使该音符延长到原来的1.5倍。音符后可带有多个圆点。例如一个四分音符后带两个圆点则演奏时间为全音符的1.5＊（1.5＊1／4）倍。圆点也可用在休止符（P）的后面。

Tn　规定速度，即每分钟四分音符的数目。n在32到225之间，默认为120。

MB　指明用“后台”方式演奏音乐。即在演奏音乐的同时，程序可做其它的工作。

MF　指明用“前台”方式演奏音乐。即：执行完PLAY语句之前不执行程序中后续的语句。这是默认的演奏方式。

MN　正常演奏。即每个音符演奏由L命令指定时间的7/8。这是默认的演奏方式。

ML　连续演奏。每个音符按由L命令指定的时间演奏。

MS　断续演奏。每个音符演奏由L命令指定时间的3/4。

Turbo Basic中，音乐缓冲区中默认可放32个音乐符，可以用$SOUND伪语句扩充音乐缓冲区的大小。有关后台音乐演奏方式的用法，请参见ON PLAY语句。

通常在PLAY语句中处理的是字符串表达式的值：音符是字符串，命令是字符串，各种n参数也是以数字形式出现的字符串。如果想在PLAY语句中引用变量，则需使用下面的“X”命令和“=”命令：

X　VARPTR$ (s$) 用于引用字符串变量s$。例如：

```
a$ = "CDE"
PLAY "EDE X "+VARPTR$ (a$)
```

与

```
PLAY "EDECDE"
```

效果一样。

=　VARPTR$ (Y) 用于引用数值变量Y。例如：

```
FOR n=1 TO 3
    PLAY "O=+VARPTR$ (n) +"GGG"
NEXT n
```

与

```
PLAY "O 1 GGG"
PLAY "0 2 GGG"
PLAY " O 3 GGG"
```

的效果一样。

注意：**PLAY语句处理的是字符串表达式，可直接引用字符串变量而不用X VARPTR$ (s$) 的格式。**

举例：

```
S1$ = "O3 L4 C L2 EE L8 E L4 EEDCF L1 <A"
S2$ = "O3 L8 CCC L4 ED L2 <A L4 A L1 >C"
```

```
S3$="L4 C L10 FF F L1 E"
PLAY "T150"            ' speed up tempo
FOR N=1 TO 4
    PLAY S1$
NEXT N
PLAY S2$+S3$
END
```

PMAP

功能: 图形屏幕的物理坐标与实用坐标的相互转换

格式: PMAP(i, option)

注释: PMAP 函数提供了在两种坐标系之间互相转化的手段。i是要进行映射（转变）的坐标。option 是 0 到 3 之间的一个整数，由它表明 i 是 x 坐标还是 Y 坐标，并表明要进行哪种映射，其具体含义如:

0　把实用坐标的 X，映射为物理坐标的 X。

1　把实用坐标的 Y，映射为物理坐标的 Y。

2　把物理坐标的 X，映射为实用坐标的 X。

3　把物理坐标的 Y，映射为实用坐标的 Y。

POINT

功能: 返回某象素的颜色／获取前一个点（LPR）的坐标

格式: POINT(X,Y)

或:

POINT(n)

注释: 第一种格式返回点（X，Y）的颜色号；如果X或Y超出范围，返回—1。

第二种格式根据n的值求前一个点（LPR）的坐标值。n与返回值的关系如下:

n	返回值
0	物理坐标 X
1	物理坐标 Y
2	实用坐标 X
3	实用坐标 Y

举例: 下面的程序在中分辩率图形方式下画一个圆，并用1号颜色画出圆心，再显示圆心点的颜色号，最后显示圆心的物理坐标 X 和 Y 的值。

```
SCREEN 1
CIRCLE(160, 100) ,50,2
PSET(160, 100), 1
PRINT POINT (160, 100)
PRINT POINT (0), POINT(1)
```

POKE

功能:　往指定地址写一个字节

格式:　POKE offset , n

注释:　offset 表示段内偏移地址。段地址由前面的DEF SEG 语句指定。n是要写入的内容，它是 0 到 255 之间的整数。执行 POKE 语句时不进行检测，所以此语句使用不当会搞乱系统。

POS

功能:　返回当前光标的横向位置(列数)。

格式:　POS(X)

注释:　X 是虚变元，它的值对函数返回值无影响。

PRESET

功能:　在图形屏幕上画一点。

格式:　PRESET(X, Y) [, color]

注释:　(x,y) 是图形屏幕上要画点处的坐标：它既可用绝对形式，也可用相对形式给出。

Color 是一个整数表达式，确定 (x, y) 处的颜色。缺省的颜色号与PSET 相反，屏幕的工作模式对颜色属性有限制；至于各种屏幕模式下有效的颜色范围，参见 PALETTE 语句。

PRINT

功能:　将数据送至屏幕输出。

格式:　PRINT [expression list] [;]

注释:　expression list(表达式表) 是一系列用逗号或分号分隔的数字或/和字符串表达式。如果 expression list 的末尾未用分号，Turbo Basic 在 expression list 中的信息之后输出一个回车。如果省略了表达式表，PRINT 只输出回车。

expression list 中的表达式若用逗号分隔，PRINT 在输出两个表达式中信息时，两者相隔一个制表位；而用分号分隔时，两信息连着显示。

功能: 将格式化信息送到屏幕输出

格式: PRINT USING format string; expression list [;]

注释: format string(格式字符串) 是一个字符串常数或变量,描述expression list (表达式表) 中信息的显示格式.

expression list (表达式表) 是要打印字符串或数字信息;各表达式间用逗号,空格或分号分隔.

PRINT USING 语句忽略表达式中的标点. 下面介绍格式字符串中的格式符:

1. 对字符串型数据, 有三种格式符可用于规定要显示字符串的长度:

!　只显示字符串的第一个字符.

\n\　共显示n+2个字符, 两个斜杠之间的n不是数, 而是n个空格。以左对齐的格式显示字符串。如果数据的长度小于格式字串规定的长度, 则在字符串的右边补以空格.

&　按数据本身的长度显示。

例如: A$ = "ABC" : B$-"XY

```
PRINT USING "!"; A$, B$
PRINT USING "\   \"; A$, B$
PRINT USING "&"; A$, B$
```

执行结果如下:

```
AX
ABC   XY
ABCXY
```

2. 显示数字用的格式字符串主要由若干个#号构成, 还可带有其它字符:

#　用#的个数规定显示的数字的位数, #之间可有小数点. 数字按右对齐的格式显示。如数字的位数少于格式字符串的规定, 则在左边补以空格; 如果数字超出了格式字符串的规定, 也显示这个数, 但是在数字的左边标上%, 以示警告. 例如:

```
PRINT USING "##.## "; 111, 22, 0.276
```

显示为: %111.22 , 0.28

+　格式字串的开头或末尾带有加号时, 显示的数字前面或后面将带有符号 (正号或负号) 。例如:

```
PRINT USING "+##.##"; -68.95, 2.4
```

显示为: -68.95, +2.40

-　格式字串后的减号, 使显示的负数后面带负号。例如:

```
PRINT USING "##.##-"; -68.95, 24
```

显示为: 68.95-2.40

* *	格式字串前的 * * 使数字前多余的空格被 * 填满， * * 也使规定的数位增加了两位，例如: PRINT USING "* * ###"; 2 显示为： * * 200
$ $	格式字串前的 $ $，使显示的数字前带有一个 $， $ $ 也使规定的数字位数增加两位，但在指数形式的格式字串(见下文)中不能使用 $ $，例如: PRINT USING "$ $ ###.##"; 456.78 显示为: $ 456.78
* * $	格式串前的 * * $，使显示的数字前带有一个 $，并使前面多余的空格填满 *。它也使规定的数字位数增加三位。例如: PRINT USING "####.##"; 1234.5 显示为: 1,234.60
^^^^	格式字串带 ^^^^时，用指数形式显示数据，即有效数字加上E ± nn 的形式，如: PRINT USING "#.## ^^^^"; 234.56 显示为: 2346E+01
_	格式字串的前后带有底划线时，可在格式字符串的前后再带上其它任何字符串。显示数字时，在数字的前后将显示这些字符串。例如: PRINT USING "_This is an example ##.## OK"; 75.3 显示为: This is an example 75.30 OK!

PRINT # 和 PRINT # USING

功能: 将数据写入顺序文件

格式: PRINT #filenum, [USING format string ;] expression list[;]

注释: filenum 是文件打开的序号; format string(格式字符串) 为任选项，是一系列格式字符 (在 PRINT USING 语句中描述的)。expression list(表达式表) 是一组要输出到顺序文件的数值型和/或字符串型表达式。

PSET

功能: 在图形屏幕上画一个点

格式: PSET(x,y) [, color]

注释: (x,y) 是图形屏幕上要画点处的坐标; 它既能以绝对形式，又可以相对形式给出。

color 是一个整数表达式，确定 (x,y) 上的颜色; 缺省的颜色与PRESET 语句中的相反。屏幕工作模式限定了颜色属性的范围; 在中分辨率模式下，可接收的颜

色号为 0～3；在高分辨率模式下，只允许使用 0 或 1.

PUT(文件)

功能：往随机文件中写一条记录

格式：PUT [#]filenum [，record number]

注释：filenum 是文件打开时设定的文件号.

record number（记录号）是一个1～16777216 之间的整数表达式，描绘要写入的记录。

PUT(图形)

功能：PUT(x,y)，array [,option]

注释：(x, y)一个矩形区域的左上角的坐标，数组array中存放的图形将画到此区域中；它既可以绝对形式，也可以相对形式指定.

array 是数组名，该数组中存放有图形数据.

option 是下面五个保留字之一（默认为XOR)，它说明怎样把数组中的图形画到屏幕上：

option	效果
PSET	原样不变地把数组中的映象（图形）送到屏幕上.
PRESET	把数组中的内容取反之后再送到屏幕上.
XOR	用数组中的内容与屏幕上对应象素的值进行 XOR 操作.
OR	用数组中的内容与屏幕上对应象素的值进行 OR 操作.
AND	用数组中折内容与屏幕上对应象素的值进行 AND 操作

PUT 图形语句是用数组中映象（图形及其背景）中各象素的颜色码，与屏幕上对应区域中原有象素的颜色码重新组合为新的颜色码．中分辨率方式的各种组合如下：

		数组颜色码				
		PSET	PRESET	XOR	OR	AND
屏幕颜色码		0123	0123	0123	0123	0123
	0	0123	3210	0123	0123	0000
	1	0123	3210	1032	1133	0101
	2	0123	3210	2301	2323	0022
	3	0123	3210	3210	3333	0123

从表中可以看出，选PSET 时，新的颜色码完全与数组中的一样，而与屏幕上原有的颜色码无关。这时将用数组中的信息，取代屏幕上该矩形区域中的全部信息。因此，选用 PSET 时，在把数组中的图形部分画到屏幕的同时，会改变该矩形区域上的背景 .

当要制造动画时，XOR 特别有用：第一次执行PUT XOR 可把图形送往屏幕，而在同一位置第二次执行 PRT XOR 同一对象时，将会使原来的背景恢复原样。

举例： 下面的程序在中分辩率图形方式下，先画一个小圆，以青色填充，并用GET(图形）语句把小圆所在的屏幕矩形区域存入数组 A%。清除显示后，又在屏幕中心画个大圆，填以洋红色。最后利用循环语句，使小圆从左到右穿过大圆，为避免闪烁得太快，每次显示小圆后停顿 0.1 秒。

```
DIM A$ (200) : SCREEN 1,0               ' 数组 A%的大小见 GET(图形) 语句
CIRCLE(15,15) , 6,1                     ' 画一个小圆
PAINT(15, 15), 1,1                      ' 把小圆填充成青色
GET(4,4) - (22, 22), A%                 ' 将小圆存入数组 A%中
CLS                                     ' 擦掉小圆
CIRCLE(160, 99), 50, 2                  ' 画一个大圆
PAINT (160, 99), CHR$ (&HAA), 2         ' 把大圆填充成洋红色
FOR N=1 TO 280 STEP 2                   ' 将让小圆每次移动两个点的距离
    PUT (N, 90), A%, XOR                ' 把数组 A%中存放的小圆送往屏幕
    DELAY 0.1
    PUT (N, 90), A%, XOR                ' 在同一地点将小圆擦掉
NEXT N
```

PUT $

功能： 将字符串写入二进制文件中的当前位置

格式： PUT $ [#]filenum, string expression

注释： filenum 为文件号；string expression 是要写入文件中的字符串表达式.

RANDOMIZE

功能： 随机数产生器的种子.

格式： RANDOMIZE [n]

注释： n为任意的长整型表达式．若省略n，则执行该语句时，机器将要求用户输入一个数值.

READ

功能: 把DATA 语句中的常数读给变量

格式: READ variable [, variable]…

注释: variable 是一个数字或字符串变量.

REG(函数)

功能: 读取寄存器缓冲区中的数值

格式: REG(n)

注释: n 是0～9之间的整数表达式, 代表一个寄存器缓冲区. n与寄存器的对应关系为:

n	寄存器
0	标志寄存器
1	AX
2	BX
3	CX
4	DX
5	SI
6	DI
7	BP
8	DS
9	ES

如果n为0～9之外的整数, 将产生第5号运行时错误: Illegal Function Call.

REG(语句)

功能: 为寄存器赋值

格式: REG n, value

注释: n 为0～9 之间的整数表达式; n与寄存器间的对应关系参见REG 函数.

value 是一个0～65535之间的数字变量或表达式.

要用CALL INTERRUPT 语句调用系统的中断时, 需要先用REG 语句为有关的寄存器缓冲区赋初值; 中断调用返回后, 可用 REG 函数获得寄存器缓冲区中的结果.

REM

功能:　为程序注释

格式:　REM comment

注释:　comment(注释）是任意的字符序列．与解释型BASIC 对注释的处理不同，Turbo Basic 的 REM(注释）部分只出现源程序之中，对程序的运行毫无影响．

RESET

功能:　关闭并清除所有的文件．

格式:　RESET

注释:　RESET 与不带变元的CLOSE 语句完全相同．

RESTORE

功能:　允许重新读取DATA 语句中的常数．

格式:　RESTORE [label]

注释:　label(标号）是任选项，标识下一条READ 语句将访问的DATA 语句，如果省略 label，则使下一条 READ 语句访问程序块中第一条 DATA 语句．

RESUME

功能:　在出错处理程序段中指定返回地址．

格式:　RESUME [{0 | NEXT | label }]

注释:　RESUME 和RESUME 0 使程序回到发生错误的语句处．

RESUME NEXT 使程序返回到发生错误语句的下一条语句处．

RESUME label 使程序返回到label(标号）标识的语句处．

RETURN

功能:　从子程序中返回调用程序．

格式:　RETURN [label]

注释:　label(标号）标识用户指定程序流程"流"向的程序行．无标号时，RETURN 语句中止子程序的运行，并将程序流程直接返回到 GOSUB 语句之后．

RIGHT $

功能:　从字符串的右边开始提起子字符串．

格式:　RIGNHT $　(string expression , n)

注释:　string expression(字符串表达式）是提供字符的"母"字符串；n 为提取的字符个

数；n 的数值范围为 0～32767．如果 n 大于 string expression 的长度，则返回 string expression 本身．

RMDIR

功能:　删除子目录（类似于DOS的命令）

格式:　RMDIR path

注释:　path 是一个标准的路径描述字符串．

RND

功能:　产生随机数

格式:　RND[(n)]

注释:　n为任意整数，它的值对函数返回值没有影响。n为0时有特殊意义，返回上一个随机数。RND 的返回值是 0～1 之间的、均匀分配的随机数。

RSET

功能:　将字符型数据移入随机文件缓冲区．

格式:　RSET field variable = string expression

注释:　RSET 和 LSET 两语句均可将字符串数据放入 field variable(字段变量）；它已在前面 FIELD 语句中定义为随机文件缓冲区的相关量．

如果string expression(字符串表达式）的长度小于FIELD 语句中指定的字段变量长度，那么 RSET 通过添加空格使字段变量右对齐．这就是说，在 string expression 首字符的前面填入空格，以使 RSET 操作之后，field variable 的长度一直等于 FIELD 语句中定义的宽度．

LSET 语句用增添空格的方法使字段变量左对齐（在string expression 最后一个字符之后附加空格)．

此外，RSET 和LSET 语句也可用于向屏幕或打印机输出数据。

举例:　下面的例子用RSET 使显示的字符串采取右对齐的格式:

```
A$ = SPACE$ (20)
RSET A$ = "Right-just"
PRINT A$
```

RUN

功能:　运行程序

格式:　RUN [filespec]

注释:　不带变元的RUN 语句将重新执行该程序本身，将全部数值变量置为0，字符串变

量置为空。

如果含有filespec(文件名)，那么它是一个表示.TBC或.EXE文件（通过设定编译器 Options 选单下 Compile to 中的 Chain[链接]或 EXE 选项而生成）的字符串表达式。filespec 未包括扩展名时，自动设为.TBC；倘若 filespec 含有扩展名，那么就使用其中的扩展名。假如要运行或链接一个.EXE 文件，则必须将扩展名．.EXE 明确地附到文件名上。

SCREEN(函数)

功能：　返回屏幕上指定字符的ASCII码或字符属性

格式：　SCREEN(row，column [，option])

注释：　该函数可得到屏幕上指定位置处字符的信息。省略option 项，得到的是指定位置上字符的 ASCII 码；如果 option 项的值为 1(非零)，此函数得到的是该字符的属性(参见文本方式的 COLOR 语句)。option 项只在文本方式下有效。

在图形方式下，如果指定位置不是字符，将得到0。如果想得到指定"点"的信息，可用 POINT 函数。

SCREEN (语句)

功能：　设置屏幕模式

格式：　SCREEN [mode] [，[colorflag]] [，[apage]] [，[vpage]]

注释：　mode 为0～10之间的整数表达式，其意义如下：

mode	效　　果
0	设置为文本方式，而且不改变屏幕的宽度。
1	设置为中分辨率图形方式，并将屏幕改变为 40 个字符宽。
2	设置为高分辨率图形方式，并将屏幕改变为 80 个字符宽。
7	在 EGA 上设置中分辨率方式，允许有 16 种颜色和 640 * 200 个象素。
8	在 EGA 上设置高分辨率方式，允许有 16 种颜色和 640 * 200 个象素。
9	在 EGA 上设置增强型的高分辨率图形方式，可以有 64 种调色板，每种允许有 4 或 16 种颜色（取决于 EGA 的存储器大小）和 640 * 350 个象素。
10	为具有 EGA 的单色显示器设置高分辨率方式。允许 4 种属性和 640 * 350 个象素。

colorflag(颜色标志)：　此项控制着颜色信息是否被送到显示器去，它的值是一个数值。在文本方式下，假值（0）将关掉彩色。在中分辨

率图形方式下，假值（0）将打开彩色（非0值为真）.

apage(工作页)：值在0–7之间．此项控制PRINT 等屏幕输出语句将向哪一页输出．此项仅对具有彩色图形（CGA）或增强型图形适配器(EGA)，而且屏幕方式为 0,7,8,9,10 的情况有效.

vpage(可见页)：值在0到7之间，此项用于选哪一页为可见到的页面．此项仅对 CGA、EGA 而且屏幕方式为 0,7,8,9,.10 时有效．通常省略此项，这时工作页和可见页为同一页。

SCREEN 语句执行时先清除屏幕，将前景色置为白色，背景色和边框色置为黑色，并按指定的参数为后续的语句设置屏幕显示方式．如果 SCREEN 语句中指定的方式与现行方式相同，则不发生变化.

如果是文本方式而且只指定了apage 和vpage 项，那么显示页将作相应的调整.

举例:

```
SCREEN 0, 1, 0, 0      ' 选择彩色的文本方式，工作页和可见页均为第 0 页.
SCREEN ,, 1, 2         ' 方式和颜色不变，第一页为工作页，第二页为可见页.
SCREEN 1, 0            ' 设置为中分辨率图形方式，允许彩色.
SCREEN 2               ' 设置为高分辨率图形方式.
```

SEEK

功能: 在二进制文件中为GET $ 和PUT $ 语句定位.

格式: SEEK [#]filespec, position

注释: filespec 是文件号: position 为字节号．SEEK 语句指明下一条操作第filespec 号文件的 GET $ 或 PUT $ 语句将作用于文件中的第 position 个字节处．当然，第 filespee 号文件必须已按二进制方式打开.

SELECT

功能: 根据对一个表达式测试的结果，选择一组语句执行

格式:

```
SELECT CASE expression
    CASE test list
        语句组
    [CASE test list
        语句组]
    [CASE test list
        语句组]
    [CASE  ELSE
        语句组]
```

END SELECT

注释: expression 是一个表达式，它既可以是数值型的，又可以是字符串型的.

test list 是对expression 的值进行的一系列测试，各测试项间用逗号分隔. 在SELECT 结构块中，expression 的类型和各 test list 中值的必须一致.

这些条件可以等于（不等于、大于、小于）某个值，或者是否在某一个范围内. 在一个 CASE 子句的测试表中，各个测试项之间的关系隐含为 OR 的关系，即只要有一项测试结果为真，就执行这个 CASE 子句与下一个 CASE 子句之间的语句组. 此后，跳出 SELECT 结构执行 END 之后的语句. 如果各 CASE 的条件都不成立，就执行 CASE ELSE 后的语句组.

举例: 下面举出一些CASE 子句进行的测试情况:

```
CASE  < b              ' 表达式（expression）的值小于 b 的值时为真.
CASE  14               ' 表达式的值等于 14 时为真.
CASE  b TO 99          ' 表达式值在 b 与 99 之间时为真（对一个范围进行测试).
CASE  14, b            ' 等于 14 或等于 b 时为真（对两个条件进行测试).
CASE  b TO 99, 14      ' 在 b 与 99 之间或等于 14 时为真.
```

SGN

功能: 符号（判断）函数

格式: SGN (n)

注释: n为任意的数字表达式. 当n>0时，函数返回1; n=0时，返回0; n<0时，返回 -1.

SHARED

功能: 在过程或函数中定义全局变量

格式: SHARED variable list

注释: SHARED 语句将variable list(变量表) 中用逗号分隔的各变量定义为整个程序的全局变量，而不限于它所在的函数或过程之中. 这样，不必将它们当作参数传递，函数或过程便可访问这些变量. SHARED 语句只可能出现在函数或过程定义体中，而且必须在相应定义体中任何可执行语句之前出现.

SHELL

功能: 装入并执行子过程 (child process)

格式: SHELL [parameter string]

注释: Parameter string(参数字符串）是一个字符串表达式，其中含有要运行的 .COM.EXE 或.BAT 程序("子过程") 名字，以及希望传递给这些程序的参数.

省略 parameter string 时，将出现 DOS 提示符，此时可输入 DOS 命令。然后，用 EXIT 命令可从 DOS 状态退回原程序。如果含有 parameter string，就先执行指定的程序，然后再执行 SHELL 之后的语句。

举例:

```
SHELL "DIR / W"
PRINT "Now back in Turbo Basic"
END
```

SIN

功能: 返回正弦值。

格式: SIN(X)

注释: X为数值表达式，代表要计算的角度（以弧度为单位）。

SOUND

功能: 产生指定频率和时间的声音。

格式: SOUND frequency, duration

注释: frequency 表示声音的频率，取值范围在37到32767之间，单位是赫兹。duration 是浮点数表达式，它表示该声音持续的时间，以 1 / 18.2 秒为单位。

SPACE $

功能: 返回指定长度的空格字符串

格式: SPACE $ (n)

注释: n为整数表达式；其数值范围为0～65535。SPACE $ 生成由n个空格组成的字符串。

SPC

功能: 跳过若干个空格

格式: PRINT SPC (n)

注释: PRINT SPC (n) 在打印时跳过n个空格；与TAB 函数一样，它只能在PRINT 等语句中使用。n 为 0～65535 间的整数表达式。

SQR

功能: 返回一数值的平方根

格式: SQR(n)

注释: n为大于或等于0 的数字表达式。

STATIC

功能: 在函数或过程中定义静态变量

格式: STATIC variable list

注释: STATIC 语句只能在过程或函数中定义；它将variable list(变量表) 中用逗号分隔的变量定义为具有固定内存地址，而不是堆栈地址的静态变量。静态变量的说明符只由它所在的过程或函数识别；这就是说，程序中过程或函数外部的变量可以与其内部的静态变量同名。

STICK

功能: 返回游戏棒的位置

格式: STICK(n)

注释: n为0～之间的整数表达式；其意义如下:

n	返回值
0	游戏棒 A 的 X 坐标
1	游戏棒 A 的 Y 坐标
2	游戏棒 B 的 X 坐标
3	游戏棒 B 的 Y 坐标

使用时,必须先执行一个STICK (0) 使之开始工作。

STOP

功能: 结束程序

格式: STOP

注释: 其作用与END一样。

STR $

功能: 将数值转化为字符串

格式: STR $ (n)

注释:　n为数字表达式。函数返回值为按n的正常显示形式构成的字符串.

STRIG(函数)

功能:　返回游戏棒的按钮状态.

格式:　STRIG (n)

注释:　n为0～7之间的整数表达式，其意义如下:

n	意　　义
0	自上次调用 STRIG (0) 函数以来，按了游戏棒 A 的按钮 1，函数值为-1，没按则为 0.
1	现在按下游戏棒 A 的按钮 1，则函数值为-1，没按则为 0.
2	自上次调用 STRIG (2) 的函数以后，按了游戏棒 B 的按钮 1，函数值为-1，没按则为 0.
3	现在按下游戏棒 B 的按钮 1，则函数值为-1，没按则为 0。
4	自上次调用 STRIG (4) 函数以后，按了游戏棒 A 的按钮 2，函数值为-1，没按则为 0.
5	现在按下游戏棒 A 的按钮 2，则函数值为-1，没按则为 0。
6	自上次调用 STRIG (6) 函数以来，按了游戏棒 B 的按钮 2，函数值为-1，没按则为 0.
7	现在按下游戏棒 B 的按钮 2，则函数值为-1，没按则为 0.

STRIG (语句)

功能:　控制游戏棒事件的检查

格式:　STRIG {ON |OFF |STOP}

注释:　STRIG ON ，STRIG OFF 和STRIG STOP 分别使读出游戏棒按钮状态的功能生效，失效或推迟.

STRING $

功能: 返回若干个相同字符给成的字符串。

格式: STRING $ (n, x)

或:

STRING $ (n, x $)

注释: n为字符个数; x为组成字符串所用字符的ASCII码。

对于第二种格式, 函数返回值为n个x $ 的首字符所组成的字符串。

SUB / END SUB

功能: 定义一个过程 (子程序)

格式:

```
SUB identifier [ (parameter list ) ]
      [LOCAL variable list ]
      [STATIC Variable list]
      [SHARED Variable list]
          :      语句组
      [EXIT SUB]
END SUB
```

注释: identifier(过程标识符) 是与过程相联系的唯一名称, 并且必须符合标号和变量的命名约定。

parameter list(参数表) 是任选项, 代表一组用逗号分隔的形式参数. 变元表中的参数只用来定义本过程, 与程序中其它同名变量没有关系.

SUB INLINE / END SUB

功能: 定义一个内部过程

格式:

```
SUB procname INLINE
      $ INLINE bytelist
      $ INLINE "COM File Name"
END SUB
```

注释: procname 是内部过程名 (不是汇编程序文件名)。

byte list 是字节表, COM File Name 是一个COM 类型的程序文件名, 内部过程中只允许出现这两种形式的 $ INLINE 伪语句, 但在一个过程中可以有多个 $ INLINE 语句。

SWAP

功能: 交换两个变量的值

格式: SWAP var1, var2。

注释: var1和var2是两个相同类型的变量，如果试图交换两上不同类型的变量，那么将发生第 475 号编译错误: Type Mismatch

SYSTEM

功能: 中止程序

格式: SYSTEM

注释: SYSTEM 中止一个程序，并将控制权返回到操作系统（或Turbo Basic，取决于程序启动的环境）。其功能和 STOP，END 一样。

TAB

功能: 在打印时指定数据输出位置。

格式: TAB (n)

注释: n为数据输出位置（列号）。TAB函数必须与PRINT，LPRINT 以及PRINT#等语句联合使用，将数据在第 n 列开始输出。

TAN

功能: 返回正切值

格式: TAN(n)

注释: n为数字表达式，表示计算的角度（以弧度为单位）。

TIMER $ （语句）

功能: 设置系统时间

格式: TIMER $ = String expression

注释: string expression(字符串表达式) 是按"hh : mm : ss"格式给定的时间字符串表达式。

TIMER $ （函数）

功能: 返回系统时间

格式: TIMER $

注释: 返回值的格式为: "hh: mm: ss"; 其中hh, mm, ss, 分别代表小时，分钟和秒。

TIMER(函数)

功能: 返回自午夜零点以来所经过的秒数

格式: TIMER

注释: 返回值为长整型，常与RANDOMIZE配合使用，产生随机数:

```
RANDOMIZE TIMER
```

TIMER(语句)

功能: 控制时间陷井是否有效

格式: TIMER {ON |OFF |STOP}

注释: TIMER ON，TIMER OFF 和TIMER STOP 分别使时间陷井俘获功能有效，失效和推迟。

TRON，TROFF

功能: 控制跟踪状态

格式: TRON

TROFF

注释: TRON启动跟踪功能，TROFF关闭跟踪功能。

UBOUND

功能: 返回数组中某一维的下标上界

格式: UBOUND(array (n))

注释: array是数组名，n指定数组的某一维。

UCASE

功能: 将字符串中字母全部变成大写字母

格式: UCASE(x $)

注释: x $ 为字符串表达式。

VAL

功能: 将字符串转换为数值

格式: VAL(X $)

注释: x $ 为字符串表达式。转换时，如果遇到非法字符（0–9,o，E e之外的字符）则

停止转换，返回 x $ 中最左边的数字。

VARPTR

功能: 返回变量的偏移地址

格式: VARPTR(variable)

注释: vairable是一个数值变量或字符串变量，也可能是一个数组元素。

VARPTR 函数的值表示该变量在段内的偏移地址，段地址可由VARSEG 函数获得。地址的信息通常称作指针，例如 VARPTR（×）被称为得到"指向×的指针"。

举例:

```
B%=55
DEF SEG =FARSEG (%)
Address$ =VARPTR (b%)
PRINT PEEK(Address%)
END
```

VARPTR $

功能: 返回变量地址的字符串的形式 格式: VARPTR $ (variable)

注释: Variable是一个数值9变量或字符串或一数组元素，VARPTR $ 函数可得到一个变量的字符串形式的地址，主要用于 PLAY 语句和 DRAW 语句。

VARSEG

功能: 返回变量的段地址

格式: VARSEG(Variable)

注释: Variable是一个数值变量或一个字符串变量，也可能是一个数组元素。由于 Turbo Basic 扩展了可以使用的内存空间，所以只有和段地址一起使用才能确定一个变量的完整的地址。

此函数的值在0-65535之间，使用此函数和VARPTR函数就可以得到一个变量的物理地址，这样再用 CALL 语句调用汇编语言子程序（过程）时，就可以告诉它，要处理的变元在内存中的什么地方。

所有非数组变量都放在最长可达64K的一个单独段中，因此，在一个程序中对所有非数组变量使用 VARSEG 函数都将得到同一个值。事实上，那些要依赖于 DEF SEG 的语句，如 PEEK，PLKE 等，未使用 VARSEG 函数。

VIEW

功能:　在图形屏幕上定义一个工作区域（视口）

格式:　VIEW [[SCREEN][(x1, y1) － (x2−y2) [, c[, boundary]]]]

注释:　(x1, y1) 和[x2, y2) 分别表示窗口的左上角和右下角; c是填充窗口用的背景颜色号. boundary 是窗口边框的颜色号.

VIEW语句在图形屏幕上定义一个工作窗口, 以后的绘图语句只能在窗口内绘图而不显示落在窗口外的点, CLS 语句也只清除窗口内的内容, VIEW 语句对窗口外原有的显示没有影响, 可以用 WINDOW 语句在工作窗口内定义实用坐标; 无参数的 VIEW 语句将把整个屏幕定义为窗口.

如果VIEW语句中不带SCREEN项. 那么以后涉及的坐标点都是相对窗口左上角的; 否则是相对于屏幕左上角的.

使用SCREEN语句改变图形方式时, 将取消先前用VIEW语句进行的设置.

举例:　下面的程序利用循环语句, 顺序地在屏幕上开出七个窗口:

```
SCREEN1
FOR N=0 TO 6
    VIEW (10*(N+1), 10* (N+1) - (25* (N+1), 25* (N+1) ), N, 3
    CIRCLE(10, 10), N+1, 3-N
NEXT N
```

WAIT

功能:　暂停程序, 等待从端口port输入满足条件的位图 (bit pattern)

格式:　WAIT port, n(, m)

注释:　prot是一个整数表达式, 取值范围在0到65535之间, 代表一个硬件输入端口; n 和 m 都是 0 到 255 之间的整数表达式. WAIT 语句暂停执行程序, 等待从 port 端口输入满足指定要求的位图. 从指定端口输入的一个字节数据先和 m 作异或 (XOP) 操作, 再和 n 作与 (AND) 操作; 如果结果为零, 说明输入的位图 (该字节) 不符合要求, 重新等待该端口的输入. 如果结果不为零, 则表明输入的位图符合要求, 将执行 WAIT 语句之后的语句. m 的默认值为 0.

举例:

```
WAIT &H60: 1: PRINT "hi"
```

WHILE / WEND

功能:　建立一个在其顶部进行条件测试的循环

格式:　WHILE integer expression

　　语句组

　WEND

注释:　如果整数表达式为TRUE（等于非零），那么执行WHILE和WEND之间的所有语句，然后 Turbo Basic 转向 WHILE 语句，再次测试；若条件仍成立（结果为非 0），再次执行循环体中的语句．这个过程重复进行，直至测试表达式最终为零为止．此时程序流程转向 WEND 之后的语句．

WIDTH

功能:　设置输出的行宽

格式:　WIDTH [{device | #n},]size

注释:　device 是设备名，设备名必须要放在双引号内，可用的设备名有:

SCRN:, LPT2:, LPT3:, COM1:, COM2:

其默认值为: SCRN:,

n 是文件号，只能代表上述设备文件之一；

size表示输出行的宽度，值在0到255之间。

一般说来，WIDTH语句用于决定在输出一个回车符之前能够输出多少个字符，但对不同设备，WIDTH 的用法有所不同。下面分别讨论:

对显示器，size只能是40或80，如果是在图形方式下，WIDTH语句在改变行宽的同时也改变图形方式；例如在中分辨率下，使用 WIDTH 80 语句会进入高分辨率方式．

对打印机，打印机默认的行宽为80个字符，WIDTH语句后的LPRINT语句将立即按新的行宽打印．例如，对宽行打印机，在 WIDTH "LPT1:", 132 之后的 LPRINT 语句将按每行 132 个字符打印．将打印机的行宽定为 255 时，会使逻辑行的宽度计数器失效，永远不会插入回车符．

对通讯文件，用WIDTH指定的宽度对已打开的设备没有影响，下次打开此设备后才按照此宽度输出，通讯文件默认的行宽为 255，改变通讯文件的宽度既不改变接收缓冲区，也不改变发送缓冲区，它只是在每 size 个字符后发送一个回车符．

WINDOW

功能:　定义图形坐标系统

格式:　WINDOW [[SCREEN] (x1, y1) - (x2, y2)]

注释:　(x1, y1) 和 (x2, Y2) 是一对单精度坐标，分别表示图形显示窗口

(Viewport 的对角线上两端点在新坐标系中的值．这两个端点通常是窗口的左上角和右下角．

在Turbo Basic中，标准的图形坐标系统是以屏幕的左上角为坐标系的原点，以右下角为(319, 199)或(639, 199)，我们称之为"物理"坐标．使用 WINDOW 语句可以定义与笛卡尔坐标非常近似的实用坐标系．在 WINDOW 语句之后，有关图形操作均在实用坐标系中进行，便于函数图形的绘制．

如果WINDOW语句中省略了SCREEN项，则实用坐标系的X轴的正方向是向右的，Y 轴的正方向是向上的；如果 WINDOW 语句中带有 SCREEN 项，则 X 轴的方向不变，而 Y 轴的正方向是向下的．执行 RUN，SCREEN 或无参数的 WINDOW 语句都将取消前面定义的实用坐标系．

WRITE

功能:　往屏幕上发送带逗号定界符的数据

格式:　WRITE [expression list]

注释:　expression list(表达式表) 为一系列用逗号或分号分隔的数字和／或字符串表达式．

WRITE#

功能:　向顺序文件中输出数据

格式:　WRITE #filenum，expression list

注释:　filenum 是文件被打开时的文件号或设备号．expression list(表达式表) 为一系列用逗号或分号分隔的数字和／或字符串表达式．输出折数据带有定界符，如逗号．

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):
{
"filename": "XzEwOTU1OTA4LnppcA==",
"filename_decoded": "_10955908.zip",
"filesize": 57227347,
"md5": "17ce61ef6ba44392822b4b65e3e14af5",
"header_md5": "623e9ca402fb51dce461e7ce4d1ff20f",
"sha1": "4ad5e2c63ac901ed49edcb164dc4b5f17bb27639",
"sha256": "56cf5091720b93c1bef561e2dddcf79b7d5389730007a991dd6fe9c14f3ecc19",
"crc32": 1366910327,
"zip_password": "",
"uncompressed_size": 62202883,
"pdg_dir_name": "",
"pdg_main_pages_found": 899,
"pdg_main_pages_max": 899,
"total_pages": 917,
"total_pixels": 6438923008
}